UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

GEEK MUSIC PARTY

Afonso Filipe Basílio Manco

Trabalho orientado pelo Prof. Doutor Thibault Nicolas Langlois

TRABALHO DE PROJETO

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Engenharia de Software

2015

Agradecimentos

Chegou o momento de agradecer a todos aqueles que contribuíram com algo de útil para tornar este caminho um pouco menos árduo.

Em primeiro lugar, quero agradecer ao Professor Doutor Thibault Nicolas Langlois pela orientação, paciência, motivação e constante apoio dados a este trabalho. Não me esquecerei da confiança que, desde cedo, depositou em mim desde a atribuição do trabalho até ao seu termino. Os meus sinceros agradecimentos. Muitíssimo obrigado.

Gostaria também de agradecer à minha família, sobretudo ao meu pai, à minha mãe e às minhas irmãs por me terem apoiado em todas as decisões que tomei mesmo sabendo que por vezes não estava certo.

Por fim, um agradecimento a todos os meus amigos que sempre me deram apoio e força para a conclusão deste trabalho. A muitos deles agradeço também o envolvimento na fase de testes.

A quem me é especial.

Resumo

Torna-se cada vez mais importante inovar no que diz respeito a plataformas de entretenimento multimédia. Hoje em dia são inúmeras as formas que estão disponíveis para tornar estas plataformas mais dinâmicas tornando assim possível uma maior interação com o utilizador. A título de exemplo, refiro-me ao uso de *webcams*, *smartphones*, QR *Codes*, voz, entre outros.

É comum haver mais plataformas estáticas do que interativas porque tendem a ser de cariz mais simples. A interação com o utilizador, usando qualquer das formas supracitadas, pode estimular uma maior utilização deste tipo de plataformas. O único fator que, por vezes, pode ser impeditivo desta maior interatividade consiste na necessidade da aquisição de tais equipamentos.

No caso deste trabalho, o objetivo é a realização de uma plataforma de entretenimento multimédia, mais concretamente de um *audio player* utilizado simultaneamente por várias pessoas, com suporte a um computador e, além de ser baseado num interface gráfico clássico, será também através de cartas QR *Codes* de grandes dimensões brandidas pelas pessoas em direção à *webcam*. A característica de se poder manipular um *audio player* com recurso a QR *Codes* dá um maior entretenimento à multimédia e permite, entre outras coisas, estimular uma maior interatividade entre os possíveis utilizadores. Além das cartas, serão também alvo de estudo, outras formas de interação com o *audio player*, como por exemplo, o uso de gestos.

Palavras-chave: Plataformas de entretenimento multimédia, *audio player*, *webcam*, QR *Codes*, música.

Abstract

Innovation becomes increasingly important when we are talking about multimedia entertainment platforms. Nowadays, in order to increase the interactive user there are several ways available to make platforms more dynamic. There are examples such as webcams, smartphones, QR Codes, among others.

It is common to have more interactive than static platforms because they tend to be simpler. The interaction using any of the above forms can stimulate their use. Maybe sometimes, the acquisition of this equipment is the main reason of less use.

In this case study, the main goal is to create a multimedia entertainment platform, more specifically a computer audio player used simultaneously by multiple people.

Apart from being based on a classic graphic interface, it will also be done through QR Codes cards that will be brandished in great dimention by the people that are working through the webcam.

As it is possible to manipulate an audio player with QR Codes giving a greater multimedia interaction by users and it allows, all among things to stimulate the possibility of a greater interaction among users. There are other ways of interaction with the audio player, such as cards with predefined patterns that will also be the subject of study.

Keywords: Multimedia entertainment platforms, audio player, webcam, QR Codes, music.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Nos últimos anos, temo-nos deparado com uma crescente evolução da tecnologia. Avanços estes que modificam profundamente o nosso quotidiano. A título de exemplo, o computador pessoal, o telemóvel e a *internet* são hoje recursos que nos parecem quase imprescindíveis.

Se pararmos para pensar, alguns anos atrás não existiam diversas formas de entretenimento para um grupo de pessoas utilizando a tecnologia. Possivelmente as formas mais conhecidas vão desde a consola para jogar através de uma interface gráfica e sonora, apresentada em algum dispositivo de vídeo e áudio até aos jogos de computador utilizando, primeiramente, uma LAN (*Local Area Network*) e posteriormente a WAN (*Wide Area Network*).

Mais tarde, começaram a surgir plataformas de entretenimento um pouco mais diversificadas e utilizando, para tal, IHC (Interfaces Humano-Computador) cada vez mais multimodais. Esta alteração consiste na adaptação de comportamentos naturais do nosso quotidiano, como a linguagem verbal, linguagem gestual, a audição e visão em plataformas onde seja requerido a interação humana. Desta forma, o foco passa a estar, também no modo como esta interação é realizada ao invés de estar, em grande parte das vezes, no resultado esperado após essa interação.

Recentemente, com a massificação do uso de *smartphones*, passarmos a utilizar uma tecnologia chamada *touchscreen*. Tecnologia esta que responde a toques humanos de maneira a executar uma ação. Também é possível controlar um *smartphone* utilizando apenas a nossa voz normal. A título de exemplo, o Siri [3] é uma aplicação para o sistema operativo móvel iOS [2] que funciona como um assistente pessoal inteligente que nos ajuda, se possível, no que lhe é pedido. Para tal, utiliza um *software* de reconhecimento de discurso para realizar uma tarefa como, por exemplo, enviar mensagens, agendar reuniões, efetuar chamadas entre muitas outras funcionalidades. Além de ser possível enumerar vantagens e desvantagens em tecnologias que permitam a interação com o ser humano de formas não muito usuais, o que se revela muito interessante é que elas existem e, desde que, bem aplicadas podem trazer valor adicional ao nosso dia-a-dia.

1.1 Motivação

As plataformas de entretenimento "passivas", podem ser descritas como aquelas que o utilizador não tem muito controlo sobre o conteúdo em si enquanto as "ativas" necessitam de uma maior participação por parte do utilizador.

De modo a melhorar a forma como estas plataformas tem vindo a ser utilizadas, é fundamental que as tornemos mais dinâmicas e que estejamos motivados para o seu uso de variadas formas. Uma forma de estarmos sempre em constante interação com as tecnologias atuais, é precisamente a integração dessas tecnologias na nossa vida. O uso de *smartphones* na nossa vida aumentou mas facilmente percebemos que a finalidade inicial foi expandida a enumeras outras funcionalidades. Estas novas funcionalidades estão constantemente a ser integradas cada vez que existe uma atualização importante nesse ramo, como por exemplo a integração de câmara fotográfica, possibilidade de navegação na *internet*, GPS (*Global Positioning System*), entre outros. Inicialmente, o uso era apenas controlado com o pressionar de botões existentes. Hoje em dia, a era dos *smartphones* com táteis parece que está para ficar e apesar de ainda existirem marcas que fabricam telemóveis apenas com botões, a necessidade adaptação ao ecrã tátil é quase indispensável. Esta integração requer uma constante atualização, porque cada vez mais e num espaço de tempo cada vez menor, podemos ser surpreendidos com alguma novidade.

Como tal, o que é pretendido é perceber até que ponto o uso de QR *(Quick Response) Codes* facilita e estimula a uma maior utilização quando integrado num *audio player* onde o seu *modus operandi* seja simples e pouco incómodo para quem o utiliza.

1.2 Objetivos

O objetivo deste projeto é a realização de uma plataforma de entretimento multimédia. A aplicação pode ser descrita como um *audio player* operado por várias pessoas simultaneamente no contexto de festa privada (numa sala de dimensão média com um computador).

A interação além de ser baseada num interface gráfico clássico será também orientada para reconhecer QR *Codes*. Estes, terão de ser exportados (como descrito no paragrafo seguinte) e impressos a partir da aplicação para depois serem brandidos em direção à *webcam* para que possam ser reconhecidos.

Existem duas exportações possíveis: música e *tag*. A primeira "exporta" uma música enquanto a segunda "exporta" uma *tag*. Cada música pode estar associada a um conjunto de *tags*. Assim, será possível, por exemplo, exportar uma *tag* "*rock*" e quando esta *tag*, representada pelo seu QR *Code*, for brandida em direção à *webcam* o *player* deverá reproduzir todas as musicas que tiverem associadas a si esta *tag* "rock". O mesmo se aplicará quando um QR *Code* associado a uma música for brandido em direção à *webcam* mas neste caso será apenas reproduzida a música em questão.

Com esta alteração no modo de interação, espera-se estimular uma maior utilização da

aplicação por parte de várias pessoas e, com isso, atrair aqueles que à partida não tinham intenções de participar na seleção das musicas a serem reproduzidas.

1.3 Contribuições

No início deste projeto, foi necessário realizar uma investigação aprofundada do mercado de plataformas de entretenimento existentes. Investigação essa que teve como principal objetivo ajudar a escolher o modo de interação com o *audio player* a ser implementado.

A decisão tomada foi de usar QR *Codes* porque existem diversas formas de se utilizar e ainda não existe grande aplicabilidade do mesmo quando se fala de plataformas de entretenimento. Após essa decisão, foi necessário escolher o tipo de linguagem e o tipo de aplicação. Sobre a linguagem, Java foi a primeira e única escolha porque é uma linguagem bastante usual e que permite um desenvolvimento adequado ao que é pretendido e sobre o tipo de aplicação a escolha foi uma aplicação *web*, contudo só após uma série de tentativas de desenvolvimento em no *widget toolkit* Swing é que se optou pela opção escolhida porque, além de ser mais usual, permite uma maior flexibilidade a nível de escalabilidade. Escolhidas as tecnologias a serem utilizadas foi então desenvolvido o *audio player* que suportasse, no mínimo, o formato de musicas *Moving Picture Experts Group Layer*-3 *Audio* (MP3) e *Free Lossless Audio Codec* (FLAC). Como funcionalidades principais além das tradicionais de qualquer reprodutor de áudio, é possível reproduzir determinada música aquando do brandimento de uma carta QR *Code* que foi exportada, e previamente associada, para tal. O uso de *tags* permite assim associar um conjunto de musicas a uma *tag* facilitando dessa forma uma procura por *tag* e permitindo também uma reprodução de um conjunto de *tags* quando mostrada a carta QR *Code* também previamente associada para tal.

A decisão de se utilizar QR *Codes* teve em conta algumas caraterísticas chave que serão agora descritas:

- Ser diferente - A inovação e criatividade é sempre importante quando se pretende criar algo. Até ao momento, não existe nenhum *audio player* disponível no mercado que reproduza musicas quando se brande uma carta QR *Code* em direção à *webcam* ou qualquer dispositivo que permite reconhecer um QR *Code*.

- Simplicidade - o processo de aprendizagem do uso de QR *Codes* é quase automático e espera-se dessa forma cativar novos utilizadores para o *audio player*.

- Curva de aprendizagem - os resultados esperados devem ser atingidos num curtíssimo espaço de tempo (quase instantâneo) e, como tal, a curva de aprendizagem tem de ser baixa. Neste caso, o modo de utilização é bastante simples, uma vez que é apenas necessário brandir o QR *Code* em direção à *webcam*.

- Curta interação - a par dos dois últimos pontos, o tempo de interação de um utilizador é reduzido por forma a estimular um maior número de utilizadores. Neste caso, tal como descrito na curva de aprendizagem, a interação de um utilizador pode chegar a ser mínima uma vez que apenas é necessário brandir o QR *Code* em direção à *webcam*, algo que pode demorar um segundo ou alguns segundos mais, dependendo da luminosidade, distancia da *webcam* ao QR *Code*, complexidade do QR *Code* mas esta ultima deve ser praticamente ignorada porque uma *tag* é algo que deve ser o mais simples possível e sendo simples o QR *Code* gerado também é simples.

1.4 Estrutura do documento

Este documento está organizado da seguinte forma:

- Capítulo 3 – São referenciados alguns projetos existentes no mercado que servem como modelo e que podem ser comparados a este *audio player* porque também utilizam diferentes formas de interação.

- Capítulo 4 – É descrita toda a análise que foi realizada para poder levar o projeto a "bom porto". Análise esta que não se cinge apenas ao entretenimento em si mas que também aborda as condições necessárias para a execução do mesmo.
  Além das opções tomadas foram ainda apresentadas algumas das opções que foram analisadas mas que foram descartadas apresentando, para tal, o motivo pelo qual não foram escolhidas.

- Capítulo 5 – Este capítulo tem como objetivo descrever todo um conjunto de soluções que foram tomadas para desenhar a arquitetura da aplicação e que permitiram o desenvolvimento deste projeto.

- Capítulo 6 – Apresentação, com detalhe, de toda a implementação que foi efetuada no decorrer da realização do projeto Geek Music Player. Serão apresentados alguns exemplos das opções técnicas tomadas.

- Capítulo 7 – São aqui apresentados os resultados obtidos de todo o projeto. Serão apresentados ambientes de teste utilizando o *audio player* e, adicionalmente, algumas das condições propostas para a execução do mesmo.

- Capítulo 8 – Este capítulo apresenta algumas linhas que podem ser seguidas para completar e diversificar o projeto que foi realizado.

- Capítulo 9 – Neste capitulo será apresentada a conclusão deste projeto. Esta conclusão tem como principal fundamento explicar qual é o resultado final obtido após

se utilizar este *audio player*. São realizadas também algumas comparações com o mercado de aplicações semelhantes dando alguns exemplos das vantagens de utilização do projeto em relação aos restantes.

Capítulo 2

Planeamento

O planeamento inicial deste trabalho tinha consistiu nas seguintes tarefas:

- T1 - Estudo das plataformas de entretenimento existentes.
- T2 - Estudo de formas de interação com um *audio player*
- T3 - Desenho da interface.
- T4 - Desenho do *audio player*.
- T5 - Concretização do *audio player*.
- T6 - Utilização do *audio player* em ambientes festivos simulados.
- T7 - Escrita do relatório.

Todas as tarefas estão calendarizadas na tabela 2.1. Este planeamento não foi cumprido à risca porque foi gasto algum tempo adicional na forma de interação escolhida (T2). Todas outras tarefas sofreram então uma alteração em relação ao previsto.

Tarefas	Mês/Ano
T1	Setembro e Outubro de 2014
T2	Novembro de 2014
T3	Dezembro de 2014 e Janeiro de 2015
T4	Fevereiro e Março de 2015
T5	Abril até Julho de 2015
T6	Agosto de 2015
T7	Julho, Agosto e Setembro de 2015

Tabela 2.1: Planeamento inicial do PEI.

Capítulo 3

Trabalho relacionado

Este capítulo destina-se a fazer a comparação com os estudos mais importantes dentro do contexto do projeto, que foi desenvolvido. Pretende-se assim dar uma perspetiva do estado do conhecimento até ao momento atual.

Para todos os problemas que se enfrentam é comum haver mais do que uma solução possível e, dentro deste projeto, foram várias as decisões tomadas. Sendo assim, torna-se fundamental ter o conhecimento de grande parte das alternativas disponíveis para determinada solução com o intuito de melhorar cada vez mais o seu desenvolvimento escolhendo a melhor opção disponível.

Serão então apresentados alguns projetos que tiveram por base algumas das características que se pretendeu para a realização deste projeto e em que contexto se inserem.

3.1 Sensor Music Player

Este leitor de música foi desenvolvido para dispositivos móveis que operem no sistema operativo Android. O sistema operativo Android é controlado com movimentos táteis que incidem sobre o ecrã de um *smartphone* e, tais movimentos, são a base do seu funcionamento tal como acontece com o rato e teclado quando se fala de um computador.

A forma tradicional de reproduzir uma música num *personal computer* (PC) é a seguinte: abre-se o leitor de música, escolhe-se a música e clica-se em reproduzir (normalmente com duplo clique em cima da música pretendida). Estas duas ultimas operações são realizadas com recurso ao *hardware* que temos disponível para tal e que em princípio será um teclado e rato. Num *smartphone*, até ao momento, estas mesmas operações são realizadas com recurso a movimentos com o dedo ou com uma caneta tátil, que incidem sobre o próprio ecrã. Esta analogia serve apenas para demonstrar o *modus operandi* de ambos.

A inovação do leitor de música em questão consiste no modo de interação que o utilizador tem com o *smartphone*. Neste caso, ao invés de se proceder com movimentos táteis, também conhecidos por cliques, o utilizador pode neste caso acenar com a mão à

frente do seu dispositivo, com um máximo de cinco centímetros de distância para alterar para música para a seguinte, esticar a mão à frente do dispositivo para parar a reprodução, entre outros.

Figura 3.1: Logotipo da aplicação Sensor Music Player.

3.2 TheCorpora

A aplicação TheCorpora é em muito semelhante ao Sensor Music Player mas foi desenvolvido para o sistema operativo Linux com recurso a um *robot* como *hardware*, que faz o papel de sensor. Os movimentos podem ser programados. Permite a reprodução de uma música, pausa, troca, entre outros, e estes são iniciados com os tais gestos programados e reconhecidos pelo *robot*. Embora sejam em contextos diferentes, a grande desvantagem do TheCorpora em relação ao Sensor Music Player, consiste no facto deste ter que incorporar um *robot* como *hardware* externo enquanto no primeiro o sensor vem incorporado com o *smartphone*.

Figura 3.2: *Robot* da aplicação.

3.3 Sony Vaio E Series

Neste caso, o sensor de movimentos e o *software* de reconhecimento de gestos já vem incorporado nos portáteis da série E da Sony e permite trocar de música, aumentar ou diminuir volume apenas movimentando as mãos em frente do ecrã. Estas funcionalidades são programáveis mas já estão automaticamente disponíveis com o leitor de música Windows Media Player fornecido pelo sistema operativo Windows 7.

Figura 3.3: Exemplo de um portátil Sony Vaio E Series

3.4 Aiekon

O Aiekon é um leitor multimédia que funciona orientado por comandos de voz mas que permite os controlos convencionais. Entre os comandos disponíveis, encontram-se os seguintes: reproduzir, parar, silenciar, avançar, voltar, visualizar artistas, repetir ou escolher canções aleatoriamente, ver género e nome da música. O leitor permite a criação e edição de *playlists*, permitindo assim a um utilizador especificar musicas por título, artista ou género. Neste caso, o *hardware* externo necessário para o funcionamento do leitor multimédia é um microfone, sendo que hoje em dia é comum o mesmo já vir incorporado, no caso de se tratar de um computador portátil.

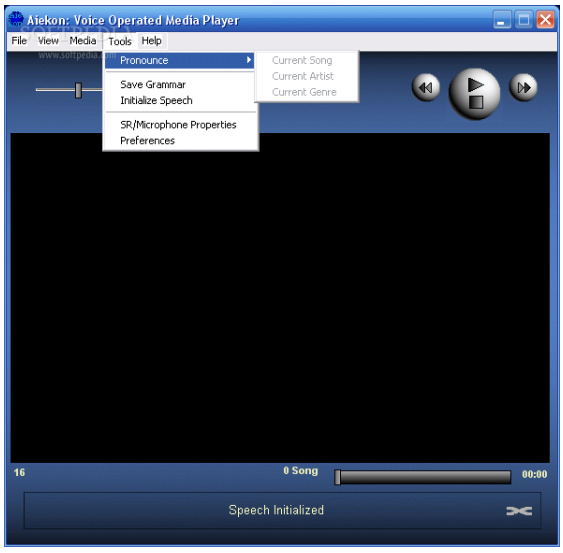


Figura 3.4: Exemplo de um ecrã do leitor Aiekon.

Capítulo 4

Análise

Para o desenvolvimento deste projeto, foi necessário efetuar um estudo detalhado das melhores bibliotecas disponíveis para a implementação do mesmo. Para a reprodução de música, foram analisadas várias possibilidades que serão descritas no presente capítulo e que terão a apreciação final da biblioteca escolhida. Em alguns casos, também serão apresentadas algumas das opções descartadas explicando o motivo pelo qual não foram escolhidas. Como tal, serão apresentadas as seguintes opções:

4.1 Reprodução de MP3

MP3 ou *Moving Picture Experts Group*-1 Layer 3 é um dos primeiros tipos de compressão de áudio com perdas embora estas sejam quase impercetíveis ao ouvido humano mas apesar disso é uma das extensões mais conhecidas e utilizadas. Foi desenvolvido em 1993 e tem embutido, em si, um *metadata*. *Metadata* é, de uma forma muito sucinta, um conjunto de atributos com dados e que permite guardar informações sobre um determinado ficheiro. Alguns exemplos dos atributos existentes aplicados a este caso são: título da música, nome do artista, ano de lançamento, imagem da capa do álbum, entre outros. A opção escolhida para reproduzir este modo de compressão de áudio chama-se Java Media Framework (JMF) embora também tenham sido alvos de análise *Freedom for Media in* Java (FMJ) e JLayer. Como tal, serão agora explicados, com um pouco mais de detalhe, as três opções analisadas.

4.1.1 Java Media Framework

Esta biblioteca, usualmente conhecida por JMF, teve a sua ultima atualização no ano 2008 após o seu lançamento em 2003. É das poucas que não tem sofrido alterações o que diz muito da sua fiabilidade. Entre várias funcionalidades existentes destaca-se pela intuitiva interface disponível e sobretudo por ser das poucas que garante a reprodução em vários sistemas operativos (Windows, Linux e Mac). Tem algumas lacunas, sobretudo na obtenção de *metadata* mas para resolver esse défice foram utilizadas outras bibliotecas

mais específicas para tal que serão detalhadas mais à frente (4.3). A versão utilizada é a 2.1.1e.

4.1.2 Freedom for Media in Java

Freedom for Media in Java ou simplesmente FMJ é uma biblioteca bastante semelhante à JMF que foi desenvolvida em 2007 e que passado um ano teve a sua última atualização. Demonstrou fiabilidade também nos mesmos sistemas operativos testados e pecou apenas por não ter uma API (*Application Programming Interface*) mais simples, como foi o caso da JMF.

4.1.3 JLayer

A biblioteca JLayer foi desenvolvida em 1999. Permite sobretudo descodificar e converter compressões em MP3. No sistema operativo Windows tem uma fiabilidade semelhante ao JMF mas quando testada em Linux ou Mac revela alguns erros na reprodução de algumas musicas. Como tal, descartou-se esta opção. A versão estudada desta biblioteca foi a 1.0.1.

4.2 Reprodução de FLAC

Apesar do MP3 se ter tornado o tipo de compressão mais utilizado ao dia de hoje não é o melhor. Como prova disso existe o FLAC. Como a própria tradução do inglês (*Free Lossless Audio Codec*) indica, este significa Codec de Áudio Livre Sem Perdas, ou seja, é um *codec* de compressão de áudio sem perda mantendo uma excelente qualidade de som.

Desenvolvido em 2003, este permite recuperar a versão original de um áudio o que não acontece no caso do MP3. Tal como no MP3, também existe um *metadata* embutido neste modo de compressão. A opção escolhida foi o BasicPlayer 3.0 e, tal como no JLayer, também foi analisada uma outra opção chamada musique. Serão detalhas também as duas opções.

4.2.1 BasicPlayer 3.0

Desenvolvido em 2004 a biblioteca BasicPlayer vai na versão 3.0 e teve a ultima atualização em 2007. Esta é muito semelhante ao JMF mas para FLAC. É bastante fácil de se utilizar fruto da sua interface bastante intuitiva. Também reproduz com fiabilidade nos mesmos sistemas operativos que o JMF. Peca apenas por arrastar consigo algumas dependências que são obrigatórias:

4.2.1.1 Tritonus Share

Desenvolvida em 1999 e atualizada pela ultima vez em 2000, esta biblioteca permite resolver reproduzir o formato FLAC. Como a sua API não é muito intuitiva e simples de utilizar esta biblioteca acaba por servir como módulo da opção escolhida. A versão é a 0.3.6.

4.2.1.2 Commons Logging API

Esta biblioteca apenas serve para fornecer uma API para efetuar o registo de aplicações baseadas em servidores. Foi desenvolvida em 2002 e atualizada pela última vez em 2014. Tem obrigatoriamente de ser consumida porque é uma dependência direta do BasicPlayer mas não acrescenta muito em termos de reprodução do formato FLAC. A versão utilizada foi a 1.2.

4.2.1.3 jFlac

Tal como a anterior, a biblioteca jFlac também tem obrigatoriamente de ser consumida e acaba por ser bastante semelhante à Tritonus Share. Foi desenvolvida em 2004 e teve em 2011 a sua ultima atualização. A versão utilizada é a 1.3.

4.2.2 musique

Além de biblioteca, musique funciona também como *audio player*. Desenvolvida em 2011, reproduz diversos formatos, entre eles o MP3. Tinha tudo para ser utilizada para reproduzir tanto MP3 como FLAC mas não se comporta tão bem no sistema operativo Mac como no Windows e Linux. Só por isso, a decisão acabou por BasicPlayer.

4.3 *Metadata*

Metadata ou traduzido à letra metadado são dados sobre outros dados. Um *metadata* contém normalmente informações sobre dados que podem ser úteis como por exemplo informação sobre outros dados. Pode parecer redundante mas exemplificado torna-se bastante mais fácil de se perceber.

Como podemos facilmente perceber pela figura 4.1, para cada música, existe um dado "*Genre*", "*Comment*", "*Artist*" entre outros que auxiliam, por exemplo, na procura de uma determinada música. Como tal, neste *audio player* desenvolvido também foi necessário extrair esta informação de cada música para se apresentar ao utilizador, mais concretamente o nome do artista ou grupo, o título da música e a imagem de capa do álbum. Para a extração de *metadata* no formato de um MP3, foram analisadas algumas bibliotecas

mas não havia nenhuma extremamente simples. Sendo esta extração de demasiada importância e não sendo demasiado complexa, decidiu-se criar uma biblioteca própria com o nome de musictagid3.

A.	Genre	File Name	Comment	Artist	Title	Track №	Album	Year
	Rock	22-Driving_Through_Kash...	Led Zeppelin (Ph...	Led Zeppelin	Driving Through Kashmir (Kashmi...	22	Physical Graffiti [D...	2015
	Rock	21-Boogie_With_Stu_(Sun...	Led Zeppelin (Ph...	Led Zeppelin	Boogie With Stu (Sunset Sound...	21	Physical Graffiti [D...	2015
	Rock	20-Everybody_Makes_It_T...	Led Zeppelin (Ph...	Led Zeppelin	Everybody Makes It Through (In T...	20	Physical Graffiti [D...	2015
	Rock	19-Houses_Of_The_Holy_(...	Led Zeppelin (Ph...	Led Zeppelin	Houses Of The Holy (Rough Mix...	19	Physical Graffiti [D...	2015
	Rock	18-In_My_Time_Of_Dying_...	Led Zeppelin (Ph...	Led Zeppelin	In My Time Of Dying (Initial / Rou...	18	Physical Graffiti [D...	2015
	Rock	17-Sick_Again_(Early_Vers...	Led Zeppelin (Ph...	Led Zeppelin	Sick Again (Early Version)	17	Physical Graffiti [D...	2015
	Rock	16-Brandy_&_Coke_(Tram...	Led Zeppelin (Ph...	Led Zeppelin	Brandy & Coke (Trampled Under...	16	Physical Graffiti [D...	2015
	Rock	15-Sick_Again.flac	Led Zeppelin (Ph...	Led Zeppelin	Sick Again	15	Physical Graffiti [D...	2015
	Rock	14-Black_Country_Woman...	Led Zeppelin (Ph...	Led Zeppelin	Black Country Woman	14	Physical Graffiti [D...	2015
	Rock	13-Boogie_With_Stu.flac	Led Zeppelin (Ph...	Led Zeppelin	Boogie With Stu	13	Physical Graffiti [D...	2015
	Rock	12-The_Wanton_Song.flac	Led Zeppelin (Ph...	Led Zeppelin	The Wanton Song	12	Physical Graffiti [D...	2015
	Rock	11-Night_Flight.flac	Led Zeppelin (Ph...	Led Zeppelin	Night Flight	11	Physical Graffiti [D...	2015
	Rock	10-Ten_Years_Gone.flac	Led Zeppelin (Ph...	Led Zeppelin	Ten Years Gone	10	Physical Graffiti [D...	2015
	Rock	09-Down_By_The_Seaside...	Led Zeppelin (Ph...	Led Zeppelin	Down By The Seaside	9	Physical Graffiti [D...	2015
	Rock	08-Bron-Yr-Aur.flac	Led Zeppelin (Ph...	Led Zeppelin	Bron-Yr-Aur	8	Physical Graffiti [D...	2015
	Rock	07-In_The_Light.flac	Led Zeppelin (Ph...	Led Zeppelin	In The Light	7	Physical Graffiti [D...	2015
	Rock	06-Kashmir.flac	Led Zeppelin (Ph...	Led Zeppelin	Kashmir	6	Physical Graffiti [D...	2015
	Rock	05-Trampled_Under_Foot....	Led Zeppelin (Ph...	Led Zeppelin	Trampled Under Foot	5	Physical Graffiti [D...	2015

Figura 4.1: Metadatos associados a um conjunto de ficheiros FLAC.

4.3.1 musictagid3

Esta biblioteca fornece uma API muito simples e fica assim disponível para reutilização em outros projetos desde que necessária. Permite extrair um *metadata* de um MP3 e para isso foi utilizado o padrão de desenho Adapter. Este padrão foi utilizado porque as bibliotecas de JAVA disponíveis para esta extração de *metadata* não tinham o contrato que era pretendido. Deste modo a solução passou por encapsular os objetos necessários, modifica-los e com isso garantir uma interface simples e compatível com o que se pretende para este projeto. Para a extração de *metadata* no formato de FLAC foi utilizado a biblioteca Jaudiotagger (4.3.3).

Foram assim, utilizadas algumas bibliotecas que ajudaram a extrair as imagens de capa do álbum e a duração de cada música tanto para um MP3 como para FLAC.

4.3.2 Jmpatric/mp3agic

Desenvolvida em 2006, esta biblioteca foi utilizada apenas para extrair a duração de uma música em formato MP3. Apesar de ter uma API bastante simples não foi utilizada porque ao extrair outro tipo de metadatos, como por exemplo a imagem, tem comportamentos diferentes operando nos três tipos de sistemas operativos já abordados. A versão utilizada é a 0.8.3.

4.3.3 JAudiotagger

Desenvolvida em 2004, esta biblioteca permite obter informações sobre ficheiros com a extensão FLAC. Devolve ainda o tempo, em segundos, de uma música em FLAC. Esta

biblioteca está apenas 75% completa [6]. Contudo, após vários testes com vários ficheiros concluiu-se que é uma biblioteca fiável. A versão em questão é a 2.0.3.

4.4 QR Code

QR *Code* é um código de barras bidimensional que pode ser convertido em texto, *Uniform Resource Identifier* (URI), número de telefone, localização georreferenciada, endereço de e-mail, contacto ou numa mensagem e que foi criado em 1994 pela marca Denso Wave Incorporated.

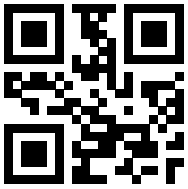


Figura 4.2: Exemplo de um QR Code com o texto "PEI FCUL".

Neste projeto, um QR *Code* é utilizado para guardar texto. Texto esse que poderá estar associado a um identificador de uma música ou um identificador de uma *tag*. Esta associação é automaticamente realizada pelo *audio player* e é aqui que existe a maior diferença em relação ao mercado de plataformas de entretenimento disponíveis.

O uso de QR *Codes* permite a qualquer utilizador que tenha capacidades motoras para brandir um QR *Code* algum controlo do *audio player*. Esta possibilidade pode ser entendida como uma abertura de uma janela para utilizadores que não saibam utilizar um computador, um *audio player* ou até mesmo para quem saiba utilizar mas que por motivos cognitivos (como no caso dos idosos com dificuldades em utilizar um rato) ou para crianças que, pelo simples facto de necessitarem apenas de brandir um QR *Code* em direção à *webcam*, podem assim controlar um *audio player* sem ter muitos conhecimentos técnicos para tal, algo que no mercado de *audio players* disponíveis é um fator novo e que diferencia dos demais porque, apesar de se saber que um QR *Code* pode permitir este tipo de controlo a nível operacional continua a ser utilizando mais num contexto de partilha de informação como por exemplo ao informar o nome de uma marca ou o acesso direto ao *website* dessa marca.

Um QR *Code* pode ser também um motivo de curiosidade para quem o utiliza. Primeiro porque ao permitir a um utilizador exportar vários QR *Code* em formato físico algo que normalmente não o fazemos porque somos mais consumidores destes e segundo porque para quem não tem um conhecimento um pouco profundo das funcionalidades de um QR *Code* pode ficar surpreendido com esta aplicabilidade do mesmo no momento de decisão de reprodução de uma música ou de um conjunto de musicas.

De modo a permitir a leitura e exportação destes mesmos QR *Codes* foi necessário analisar algumas bibliotecas para tal. Esta leitura e exportação (que pode ser entendida como geração de um QR *Code*) é realizada a nível da aplicação *web* e como tal, a analise debruçou-se sobre este tipo de bibliotecas.

Para a leitura de um QR *Code* foi utilizada a biblioteca HTML5 QR Code Reader e para geração de um QR *Code* a biblioteca para QRcode ambas em Javascript. Além destas duas bibliotecas serão ainda detalhadas as opções descartadas para leitura e geração de um QR *Code*.

4.4.1 Leitura

4.4.1.1 HTML5 QR Code Reader

Esta biblioteca foi desenvolvida em jQuery por Daniel Ward permite a leitura de um QR *Code* em *browsers* compatíveis com HTML5. É uma biblioteca bastante simples mas que cumpre os objetivos pretendidos. Fornece uma API bastante intuitiva aliado também a todos os exemplos que são apresentados.
Neste caso, para se obter o texto de um QR *Code* usando uma *webcam*, basta brandir o QR *Code* em direção da webcam e após a criação uma DIV e depois invocar a função `html5_qrcode` o resultado é guardado na variável `data` como pode ser visto na figura seguinte.

```
<div id="reader" style="width:300px;height:250px">
</div>
```

(a) Criação de uma DIV

```
$('#reader').html5_qrcode(function(data){
        // fazer o tratamento da extração
    },
    function(error){
        // mostra os erros na leitura
    }, function(videoError){
        // a webcam não pode ser iniciada
    }
);
```

(b) Invocação do método

Figura 4.3: Utilização prática da biblioteca.

Para a utilização desta biblioteca com sucesso que o *browser* suporte HTML5 e que o utilizador dê permissão ao *browser* para utilizar a *webcam*.

4.4.1.2 LazarSoft jsqrcode

Desenvolvida por Lazar Laszlo, esta é também uma biblioteca muito simples de ser utilizada. Tem uma API intuitiva e apenas foi descartada a sua utilização neste projeto por ter algumas dependências externas obrigatórias porque foi desenvolvida em Javascript.

4.4.2 Geração

4.4.2.1 QRcode

Desenvolvida em Javascript por Shim Sangmin, esta biblioteca permite gerar QR *Codes* de uma forma bastante simples. Para tal, apenas requer um *browser* que suporte HTML5. Serve perfeitamente para o que é pretendido neste projeto.

4.4.2.2 jQuery.qrcode

Lars Jung foi o autor desta biblioteca que permite gerar um QR Code. O motivo principal pelo qual foi descartada foi porque tal como LazarSoft jsqrcode, esta biblioteca utiliza uma dependência externa. A versão em questão é a 0.12.0.

4.5 Geração de uma nuvem de *tags*

A cada música pode estar associada uma ou mais *tags*, sendo que, a mesma *tag* pode estar em diferentes musicas. Com um conjunto de musicas é possível gerar uma "nuvem" que, utilizando o número de vezes em que esta *tag* ocorre nas musicas e o nome da *tag* em questão, dá uma perceção geral do número de ocorrências de cada *tag*. Para gerar esta nuvem foi necessário utilizar uma biblioteca Javascript. A biblioteca escolhida para gerar esta nuvem de palavras foi a wesomeCloud. O critério de decisão baseou-se apenas na forma automática com que esta biblioteca gera palavras com diferentes cores na nuvem, algo que também pode ser feito na jQCloud mas, para tal, é necessário algumas personalização da mesma. Foram várias as implementações estudadas e que, agora, vão ser apresentadas.

4.5.1 awesomeCloud

É uma biblioteca bastante completa que peca apenas por não ter fiabilidade quando existe uma grande variabilidade entre as ocorrências das palavras. De resto, esta biblioteca é bastante simples e personalizável, como por exemplo: quer a alterar a cor das palavras geradas, alterar o tipo de letra e alterar o rácio de rotação das palavras.
Além de jQuery, esta biblioteca necessita de um *browser* que suporte elementos canvas.

4.5.2 jQCloud

jQCloud é um *plugin* jQuery desenvolvido para desenhar nuvens de palavras com balanceamento. Este balanceamento é, precisamente, o número de ocorrências da palavra. A palavra, neste caso, é a *tag*. Quanto maior for o número de ocorrências de uma determinada *tag* maior será o tamanho que esta *tag* aparecerá na nuvem. Esta biblioteca usa HTML 5 e CSS para gerar a nuvem. A versão em questão é a 0.2.0.

4.5.3 wordcloud2

wordcloud2 é o nome desta biblioteca que permite apresentar palavras representadas numa nuvem de palavras. Apresenta as palavras de uma forma aleatória e o posicionamento destas palavras também são dispostos de aleatória, ou seja, tanto poderá ser gerado uma palavra na vertical e horizontal ou qualquer em qualquer posição entre ambas. Como ponto negativo desta biblioteca, quando as palavras não estão balanceadas, por vezes, a palavra com as ocorrências maior ou menor desaparece como acontece com a palavra "grunge" que, ao ser a palavra que ocorre mais vezes (300) dentro de uma nuvem que a segunda ocorrência tem apenas um terço (100), desaparece da nuvem gerada como pode ser visto na figura (6.8). A versão utilizada foi a 1.0.1.

Capítulo 5

Desenho

Neste capítulo, vão ser descritas as opções tomadas para o desenvolvimento do *audio player*. Será feita uma descrição do funcionamento geral do *audio player* com detalhe especial sobre cada componente necessário que permitiu um ótimo funcionamento do sistema em geral.

Além de ser apresentado o diagrama de classes que facilita o entendimento das classes principais e também porque ajuda a ter uma melhor representação da estrutura e relações existentes será detalhado qual protocolo utilizado.

Adicionalmente, será também descrito o modelo de dados existente.

A proposta da interface gráfica será apresentada para mais à frente (figura 5.6) para ser comparada com a versão final (figura 7.1).

5.1 Funcionamento geral *audio player*

O *audio player* reproduz musicas em MP3 e FLAC que estão guardadas em qualquer *drive* do sistema operativo. Estas musicas estarão representadas pelo seu *path* na base de dados e são reproduzidas quando ocorrer um evento para tal vindo do *web browser*.

Estes eventos podem ser os tradicionais utilizando, para tal, o rato e o teclado ou podem ser estimulados por um reconhecimento de cartas QR *Codes* que os utilizadores podem brandir em direção da *webcam*. A figura seguinte (5.1) ilustra um conjunto de crianças onde cada uma delas tem exportado numa carta QR *Code* o seu grupo musical preferido e que alternativamente vão brandindo a sua opção em direção da *webcam* de modo a que seja reproduzido o seu gosto musical que está representado no QR *Code*.

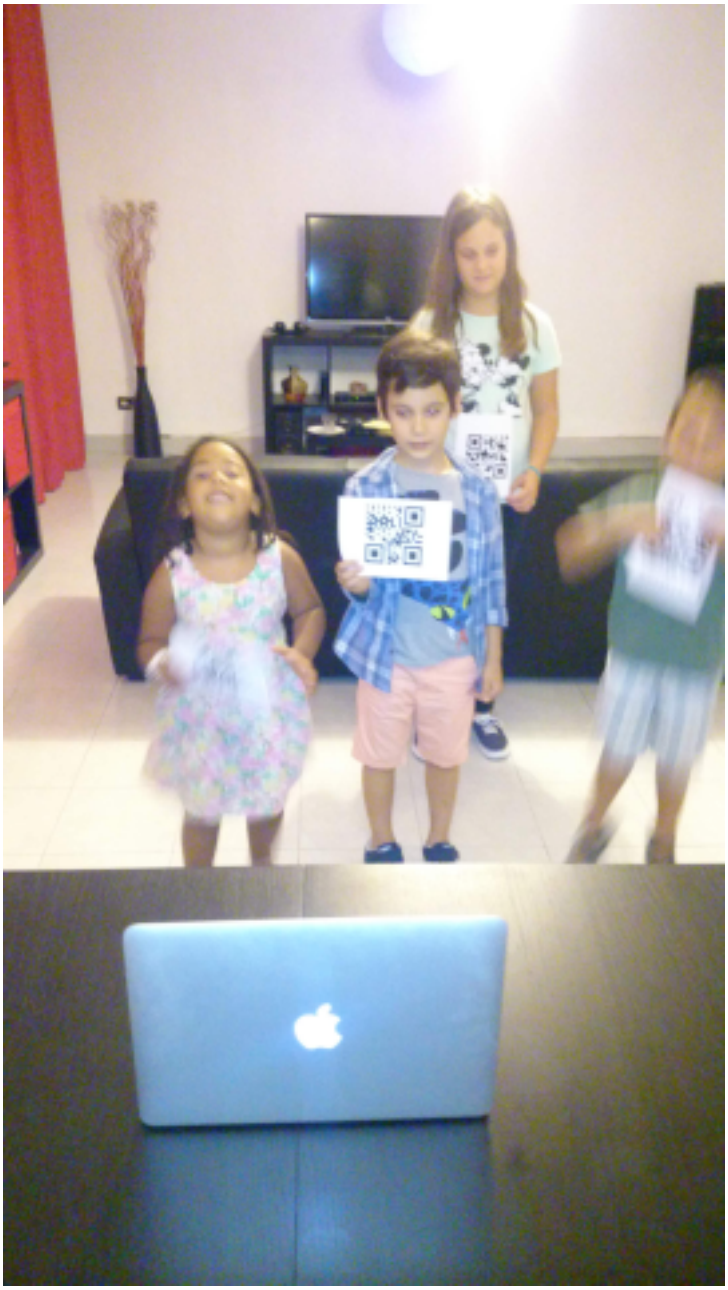


Figura 5.1: Utilização pratica de um QR *Code* por uma criança.

Como facilmente foi visto na figura anterior (5.1), cada criança tem um QR *Code* em sua posse. Este QR *Code* foi previamente exportado com a ajuda de um utilizador com algum conhecimento pratico do *audio player*. Após essa exportação, onde cada criança obteve um *QR* Code associado ao seu grupo musical preferido, cada criança teve direito a alguns minutos de "tempo de antena" onde podiam brandir o seu grupo musical escolhido representado pelo QR *Code*.
Em termos de funcionalidade, reproduzir um grupo musical ou vários, desde que exista a

mesma *tag* nas várias musicas preferidas não é nada de novo porque a criação de *playlists* é bastante usual mas ter a possibilidade de, permitir a qualquer utilizador (seja ele uma criança ou um idoso), reproduzir um conjunto de musicas a partir de um QR *Code* sem necessitar de fazer mais nada além do brandimento do QR *Code* é, sem duvida alguma, uma característica que pode estimular vários utilizadores a utilizarem o *audio player*.

Na figura 5.2 é visível um utilizador com mobilidade articular limitada e que já não tem capacidade de controlar um rato ou um teclado de forma eficiente. Contudo, neste *audio player* essa limitação não é, de todo, motivo para excluir esse utilizador do momento de decisão das musicas a serem reproduzidas. Este utilizador, apesar de limitado, conseguiu reproduzir várias vezes, sem ajuda de ninguém, algumas musicas que foram exportadas previamente.

Figura 5.2: Utilização pratica de um QR *Code* por um idoso.

Em ambos os casos (crianças e idosos), haviam diferentes fatores que faziam destes utilizadores impróprios para interação com um *audio player*. No primeiro caso, pode serem crianças e ainda não terem conhecimento para operar um *audio player* ou, no segundo caso, por estarem limitados a nível físico. Contudo, havendo a possibilidade de interação com recurso a QR *Codes* estes, sentiram-se confortáveis e divertiram-se ao reproduzir as suas musicas como pode ser comprovado pelas suas expressões faciais.

A interface com que estes utilizadores interagem, está encarregue de apresentar informações ao utilizador e interagir com o servidor de modo a que possa ser delegada um ordem de reprodução de uma música pela camada da lógica de negócio especifica para tal. Esta camada tem a responsabilidade de interagir não só com os recursos disponíveis na *drive* onde estiveram guardadas as musicas mas também tem capacidade para comunicar com a base de dados para obter informação ou atualiza-la. Após essa comunicação, a lógica de negócio volta a devolver ao servidor *web* as informações e este está encarregue de enviar estes dados para o cliente para que se possa apresentar no *web browser* de uma forma intuitiva e atraente.

A comunicação existente entre o utilizador e o *audio player* é de extrema relevância uma vez que ao se brandir cartas QR *Codes* "reconhecidas" em direção à *webcam* é automaticamente despoletado um evento que pode alterar o estado do *audio player*. A figura 5.3 pode ajudar a perceber, de uma forma mais apurada, o funcionamento geral do *audio player*.

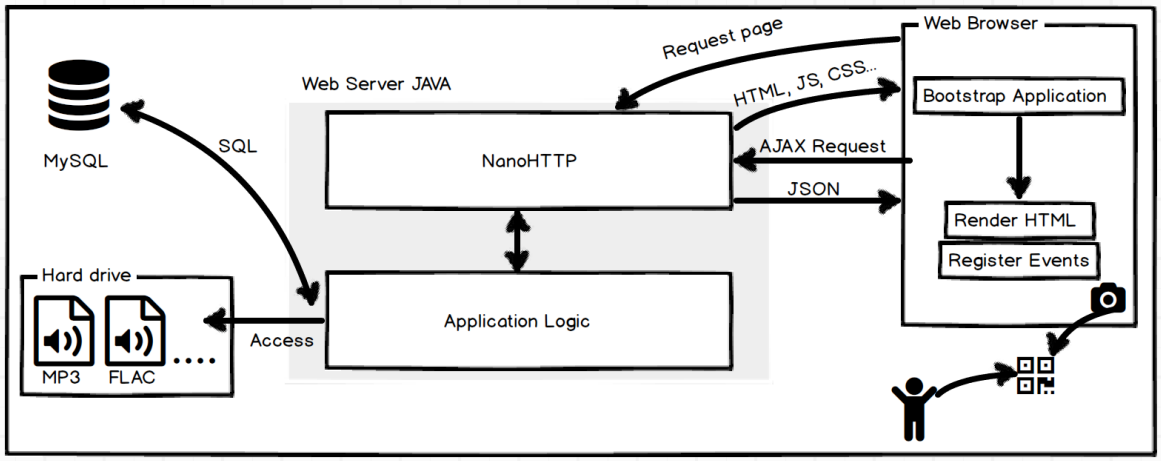


Figura 5.3: Visão geral do sistema.

5.2 Servidor *Web*

Tal como referido nas contribuições (1.3) na proposta inicial da interface não estava contemplada a inclusão de um servidor *web*. Esta solução foi adotada porque permite uma

maior flexibilidade, interatividade e dinamismo na interface gráfica uma vez que será feita com recurso a HTML5 ao invés do *widget toolkit* Swing. Este servidor *web* terá como função principal receber pedidos *Hypertext Transfer Protocol* (HTTP) por parte do cliente, neste caso o *web browser*. Pedidos estes que podem ser requisições de páginas, ficheiros ou alterações ao comportamento do *audio player*.

5.3 Protocolo

Todo o processo de acesso à camada de negócio é despoletado por um pedido HTTP realizado pelo cliente. No primeiro pedido efetuado pelo cliente ao servidor a resposta deste é o conteúdo de uma página HTML para ser visualizada no *web browser*. A partir daí, todas as respostas do servidor para o cliente vão no formato Javascript *Object Notation* (JSON) que é um padrão de comunicação baseado numa *string* que permite o tráfego de informações em baseado numa coleção de chave/valor.

Desta forma, independentemente da linguagem do cliente e do servidor, a troca de informações entre estes, ao ser realizada por JSON, permite que qualquer alteração de linguagens, não impacte o funcionamento do programa desde que continuem a respeitar o contrato definido em cada mensagem JSON.

5.3.1 Interpretação de um pedido e resposta por parte do cliente

5.3.1.1 Interpretação de um pedido

Utilizando *Asynchronous* Javascript and XML (AJAX) o cliente tem a possibilidade de "carregar" informações (enviada pelo servidor) e atualizar o estado do *audio player* sem que este tenha que voltar a construir a página *web* na integra porque é realizado de forma assíncrona.

Um exemplo deste tipo de pedidos pode ser visto nas duas figuras (5.4 e 5.5) abaixo onde primeiro é apresentada uma implementação genérica de um pedido realizado pelo cliente e na segunda uma concretização de uma requisição.

```
function ajaxSync(id, sendData) {
    $.ajax({
        type: 'POST',
        url: URL + PORT + '/' + id,
        data: sendData,
        async: true,
        dataType: 'text',
        success: function(data) {
            initapp();
            playerDispatch(data);
        }
    });
}
```

Figura 5.4: Exemplo genérico de um pedido realizado pelo cliente.

```
function getImage(id) {
        ajaxSync(id, {
        part: Cloud.getPart()
    });
}
```

Figura 5.5: Requisição de uma imagem.

5.3.1.2 Envio de uma resposta

Em Javascript, a forma de interpretar uma mensagem JSON enviada pelo servidor é bastante simples. Para tal, basta converter para a notação em JSON utilizando o método apresentado na figura 5.6 e depois o resultado obtido dessa conversão é um objeto onde cada chave/valor criada pelo servidor pode ser acedida como se de um atributo de um objeto se tratasse. A concretização pode ser visualizada na figura 5.7.

```
var object = JSON.parse(text);
```

Figura 5.6: Conversão de um resposta enviada pelo cliente.

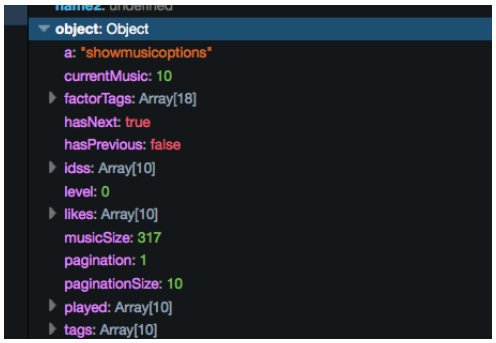


Figura 5.7: Acesso em *debug* aos atributos de uma conversão obtida.

5.3.2 Interpretação de um pedido e envio de uma resposta por parte do servidor

5.3.2.1 Interpretação de um pedido

Quando o cliente realiza um pedido, este é construindo segundo um conjunto de chave/valor que, depois, podem ser facilmente interpretados em Java. Eis um exemplo de como se pode obter o valor de uma uma chave "option".

```
// This is how the parameters passed through forms on the
// client side are accessed:
Map<String, String> params = session.getParms();
String opt = params.get("option");
```

Figura 5.8: Obtenção do valor da chave "otpion".

5.3.2.2 Envio de uma resposta

Tal como já foi referido, só a partir do segundo pedido realizado pelo cliente é que o servidor responde em notação JSON. Esta resposta é muito simples de se realizar uma vez que, para tal, foi utilizado uma biblioteca externa para criar, mais facilmente, uma notação em JSON. Esta biblioteca chama-se JSON.simple e é bastante fácil de ser utilizada. Deverá ser criado um objeto do tipo JSONObject e após isso bastará apenas fazer as atribuições chave/valor dos campos pretendidos, como se pode visualizar na imagem seguinte (5.9).

```
JSONObject json =new JSONObject();
json.put("a", "searchtag");
json.put("results", params);
```

Figura 5.9: Exemplo da atribuição de dois campos a duas chaves.

No final, bastará apenas gerar uma *string* com o resultado final que depois será interpretada pelo cliente. A imagem seguinte (5.10) mostra como é que é feita esta geração.

```
String msg = json.toJSONString();
```

Figura 5.10: Mensagem final a ser enviada ao cliente.

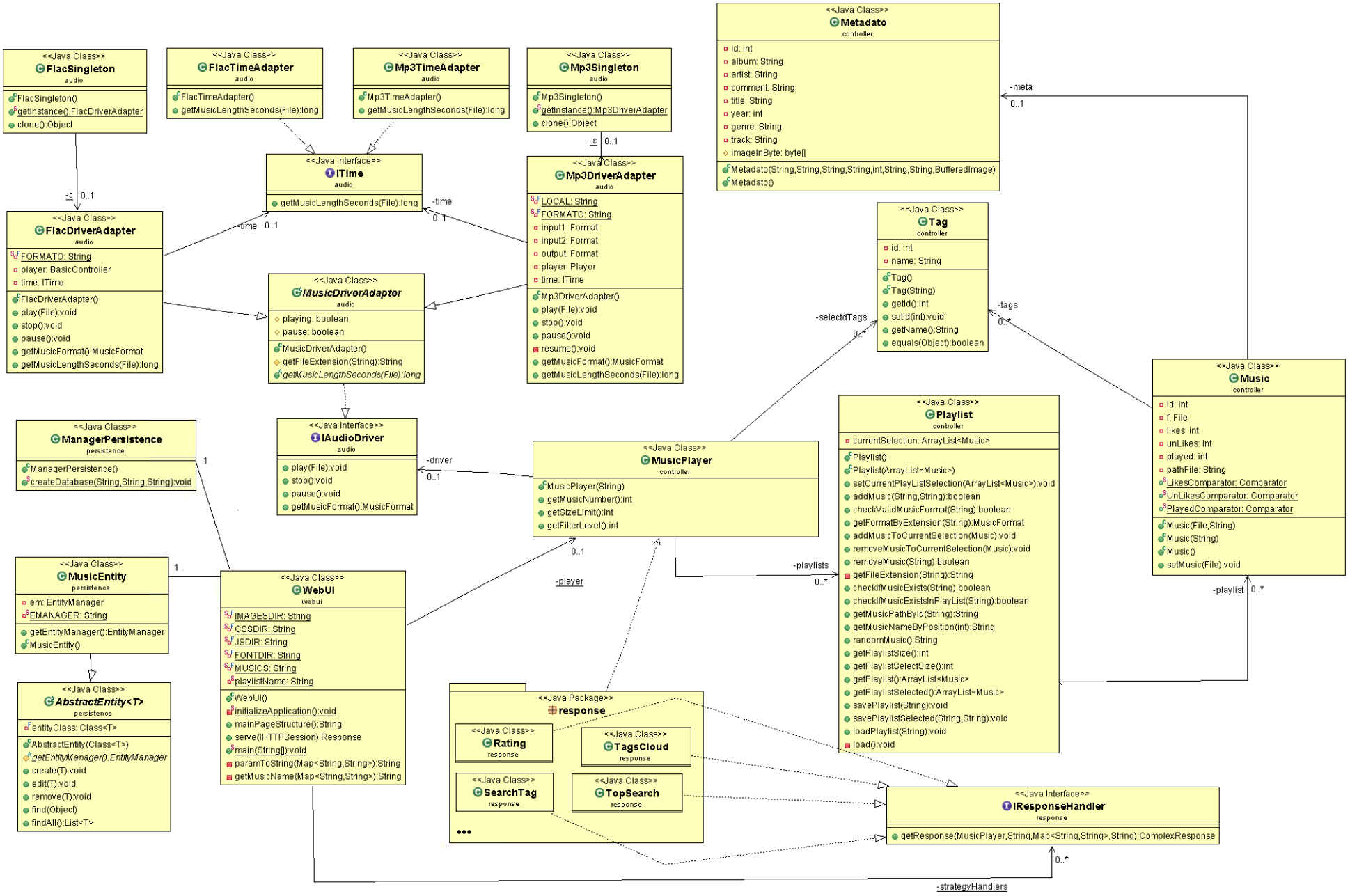
5.4 Diagrama de classes

O diagrama de classes pode ser visto na figura seguinte. A classe principal chama-se "MusicPlayer". Esta, responde aos pedidos que são enviados por parte do utilizador utilizando, para tal, as classes contidas no *package* "response". Todas as classes deste *package*

implementam a interface "IResponseHandler" e cada uma delas pode ser entendida como uma atribuição e implementação ao comportamento esperado de um pedido enviado pelo cliente. A classe "MusicPlayer" será ainda responsável por guardar os conjuntos de musicas que são representados pela classe "Playlist". "Music" é a classe que simboliza uma música e que por sua vez pode estar associadas a um conjuntos de *tags* "Tag" e obrigatoriamente a um *metadata* representado pela classe "Metadata".

Para qualquer tipo de formato musical a ser reproduzido será sempre necessário que se crie uma classe para tal. Esta classe terá obrigatoriamente que estender a classe "MusicDriverAdapter" e de implementar a interface "ITime". Neste caso, para a reprodução de MP3 e FLAC foram criadas duas classes: "MP3DriverAdapter" e "FlacDriverAdapter". A extensão da classe "MusicDriverAdapter" obriga à implementação de todos os método da interface "IAudioDriver". Esta interface estabelece o contrato necessário para, por exemplo, reproduzir uma música (método "play") ou parar uma música (método "pause"). Já a interface estabelece o contrato para a obtenção do tempo, em segundos, de cada música. Para cada classe criada deve ainda ser criado um singleton uma vez que, além de únicas, são classes que deverão ser chamadas diversas vezes.

Para armazenar os dados na base de dados é utilizado uma camada de persistência, que é representada pela classe "AbstractEntity". Para cada classe que seja necessário ser armazenada, como por exemplo é o caso da classe "Music", deverá ser criado uma classe que estenda esta classe "AbstractEntity". Para o exemplo acima supracitado a classe em questão é a "MusicEntity". A classe "ManagerPersistence" é utilizada para a criação da base de dados, caso a mesma não exista.


«Java Class» FlacSingleton audio
FlacSingleton()
getInstance():FlacDriverAdapter
clone():Object
«Java Class» FlacTimeAdapter audio
FlacTimeAdapter()
getMusicLengthSeconds(File):long
«Java Class» Mp3TimeAdapter audio
Mp3TimeAdapter()
getMusicLengthSeconds(File):long
«Java Class» Mp3Singleton audio
Mp3Singleton()
getInstance():Mp3DriverAdapter
clone():Object
«Java Interface» ITime audio
getMusicLengthSeconds(File):long
«Java Class» Mp3DriverAdapter audio
LOCAL: String
FORMATO: String
input1: Format
input2: Format
output: Format
player: Player
time: ITime
Mp3DriverAdapter()
play(File):void
stop():void
pause():void
resume():void
getMusicFormat():MusicFormat
getMusicLengthSeconds(File):long
«Java Class» FlacDriverAdapter audio
FORMATO: String
player: BasicController
time: ITime
FlacDriverAdapter()
play(File):void
stop():void
pause():void
getMusicFormat():MusicFormat
getMusicLengthSeconds(File):long
«Java Class» MusicDriverAdapter audio
playing: boolean
pause: boolean
MusicDriverAdapter()
getFileExtension(String):String
getMusicLengthSeconds(File):long
«Java Class» Metadato controller
id: int
album: String
artist: String
comment: String
title: String
year: int
genre: String
track: String
imageInByte: byte[]
Metadato(String,String,String,String,int,String,String,BufferedImage)
Metadato()
«Java Class» Tag controller
id: int
name: String
Tag()
Tag(String)
getId():int
setId(int):void
getName():String
equals(Object):boolean
«Java Class» Music controller
id: int
f: File
likes: int
unLikes: int
played: int
pathFile: String
LikesComparator: Comparator
UnLikesComparator: Comparator
PlayedComparator: Comparator
Music(File,String)
Music(String)
Music()
setMusic(File):void
«Java Class» Playlist controller
currentSelection: ArrayList<Music>
Playlist()
Playlist(ArrayList<Music>)
setCurrentPlayListSelection(ArrayList<Music>):void
addMusic(String,String):boolean
checkValidMusicFormat(String):boolean
getFormatByExtension(String):MusicFormat
addMusicToCurrentSelection(Music):void
removeMusicToCurrentSelection(Music):void
removeMusic(String):boolean
getFileExtension(String):String
checkIfMusicExists(String):boolean
checkIfMusicExistsInPlayList(String):boolean
getMusicPathById(String):String
getMusicNameByPosition(int):String
randomMusic():String
getPlaylistSize():int
getPlaylistSelectSize():int
getPlaylist():ArrayList<Music>
getPlaylistSelected():ArrayList<Music>
savePlaylist(String):void
savePlaylistSelected(String,String):void
loadPlaylist(String):void
load():void
«Java Class» MusicPlayer controller
MusicPlayer(String)
getMusicNumber():int
getSizeLimit():int
getFilterLevel():int
«Java Interface» IAudioDriver audio
play(File):void
stop():void
pause():void
getMusicFormat():MusicFormat
«Java Class» ManagerPersistence persistence
ManagerPersistence()
createDatabase(String,String,String):void
«Java Class» MusicEntity persistence
em: EntityManager
EMANAGER: String
getEntityManager():EntityManager
MusicEntity()
«Java Class» AbstractEntity<T> persistence
entityClass: Class<T>
AbstractEntity(Class<T>)
getEntityManager():EntityManager
create(T):void
edit(T):void
remove(T):void
find(Object)
findAll():List<T>
«Java Class» WebUI webui
IMAGESDIR: String
CSSDIR: String
JSDIR: String
FONTDIR: String
MUSICS: String
playlistName: String
WebUI()
initializeApplication():void
mainPageStructure():String
serve(IHTTPSession):Response
main(String[]):void
paramToString(Map<String,String>):String
getMusicName(Map<String,String>):String
«Java Package» response
«Java Class» Rating response
«Java Class» TagsCloud response
«Java Class» SearchTag response
«Java Class» TopSearch response
«Java Interface» IResponseHandler response
getResponse(MusicPlayer,String,Map<String,String>,String):ComplexResponse
-c 0..1
-time 0..1
-driver 0..1
-player
-meta 0..1
-tags 0..*
-selectdTags 0..*
-playlists 0..*
-playlist 0..*
-strategyHandlers
0..*
1

5.5 Base de dados

Todas as musicas existentes e que poderão ser adicionadas ao *audio player* terão de estar obrigatoriamente em algum local físico onde este *audio player* estiver instalado. Estas musicas, no momento em que são adicionadas serão registadas numa base de dados para haver a possibilidade de existir um conjunto de musicas disponíveis. Mas, ao invés de ser o ficheiro a ser inserido, será apenas o caminho físico onde essa música estiver alocada.

No momento da adição de uma música, deve ser possível inserir um conjunto de *tags* que ficarão, a partir desse momento, associadas à música. Adicionalmente, será ainda registado em base de dados o *metadata* de cada música.

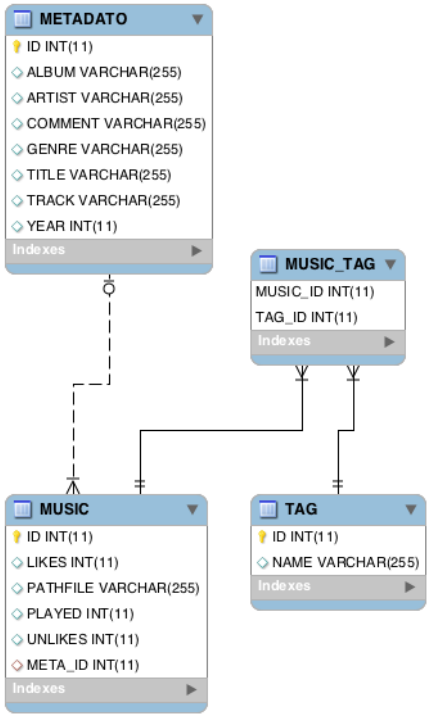


Figura 5.11: Modelo de dados do sistema.

5.6 Interface

A proposta de desenho do *audio player* foi alterada com o decorrer do projeto.

Inicialmente, este proposta, foi projetada para uma interface em JAVA e só depois de algumas experiências é que se optou por uma interface *web*.

A ideia principal foi criar uma interface bastante simples, que fosse fácil de se utilizar e de se estender. Como tal, esta interface deverá ser o mais simples possível.

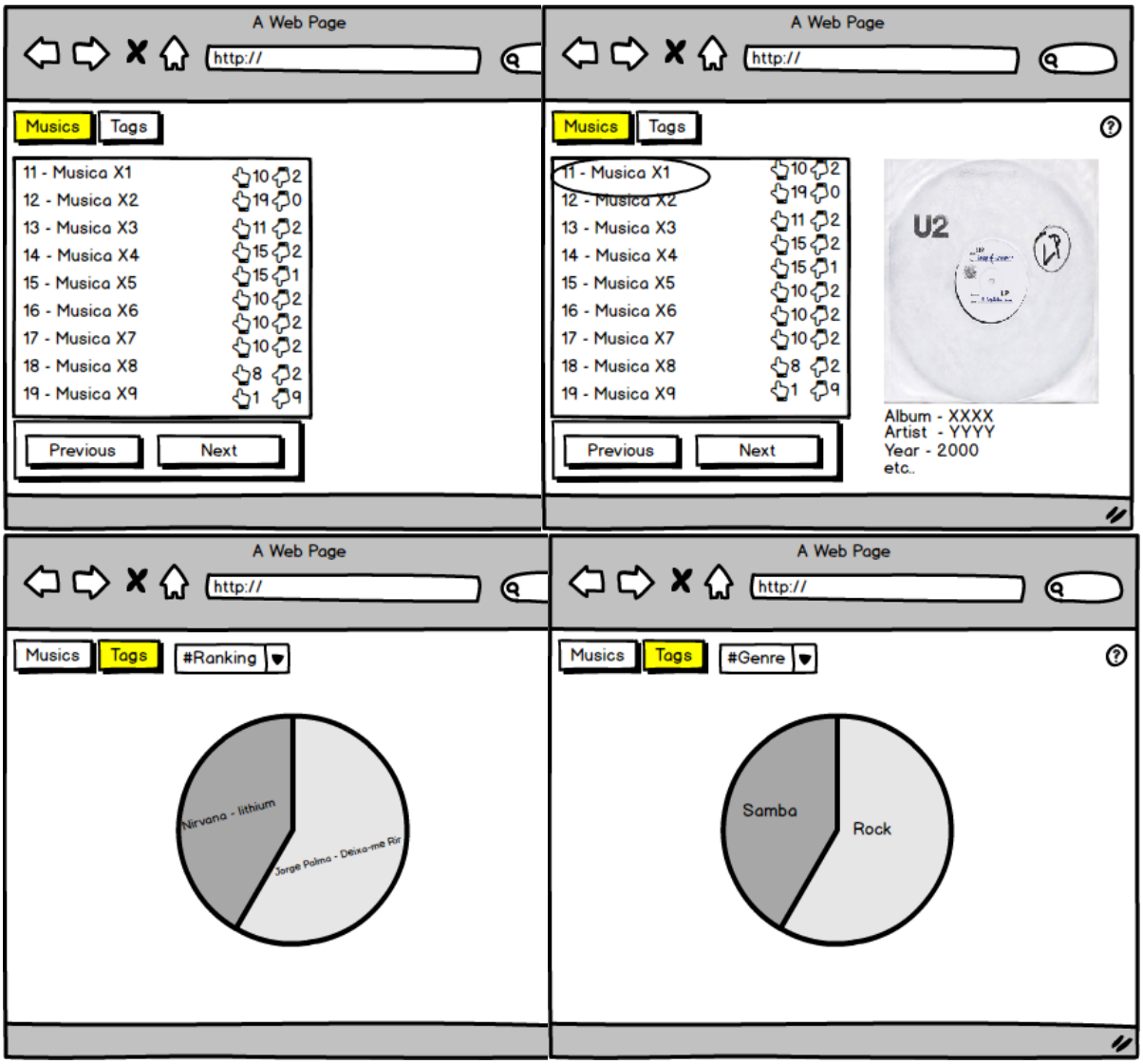


Figura 5.12: Esboços iniciais do *audio player*.

A figura(figura 5.12) apresenta alguns dos esboços que foram desenhados, antes da implementação, para se ter uma ideia do resultado final que era pretendido numa fase inicial.

Estes esboços acabaram por sofrer algumas alterações mas, num modo geral, foram muito importantes para o desenho final do *audio player*. O resultado final pode ser visto no decorrer da explicação de utilização do *audio player*. Adicionalmente será também apresentado a interface mais técnica, que, é o *audio player* tradicional executado a partir de uma linha de comandos.

Será agora descrito como se deve proceder para utilização do *audio player*.
Após se aceder ao endereço parametrizado é-nos apresentado o aspeto que pode ser visto na figura 5.13.

Do lado esquerdo, tal como pode ser visto na figura 5.14, existe um menu com opções

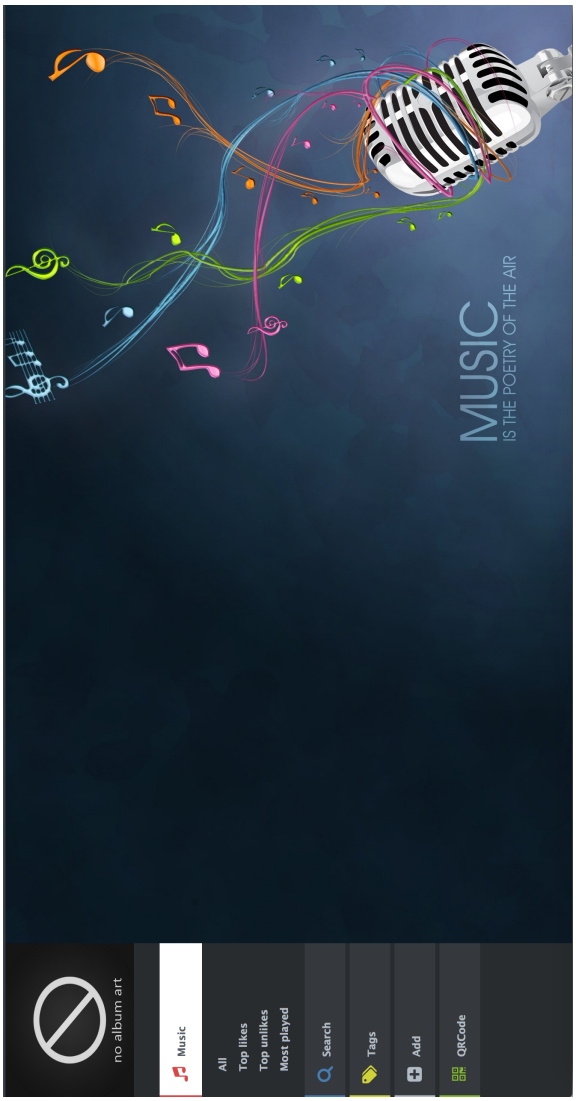


Figura 5.13: Página inicial do *audio player*.

disponíveis. A primeira opção do menu é "Music". Neste opção, ao se clicar em "All" é-nos apresentado uma lista com as musicas que podem ser paginadas como pode ser visto na figura 5.15.

Para cada música apresentada existe um conjunto de opções disponíveis: exportação, gosto e não gosto. Estas opções, podem ser visualizadas na figura 5.16, onde aparece, do lado esquerdo, um QR *Code* que, ao ser clicado, dá uma ordem de impressão dessa "música". Já do lado direito aparece as opções de gostar ou de não gostar da música. Logo a seguir ao nome da música aparece um quadrado com um número que corresponde ao número de vezes que essa música foi reproduzida.

Se a opção escolhida no menu "Music" for "Top Likes", "Top Unlikes" ou "Most Played" é-nos apresentado o mesmo listado mas ordenado pela opção escolhida.

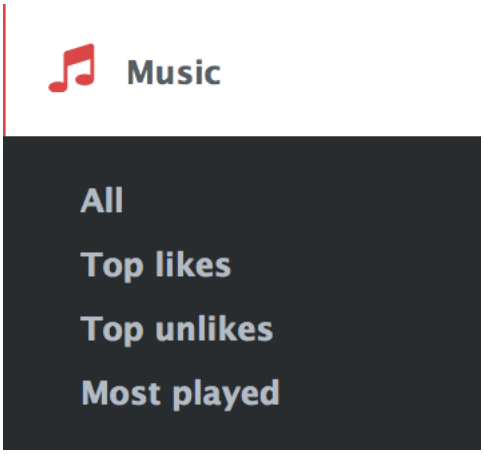


Figura 5.14: Opções dentro da opção “Music”.

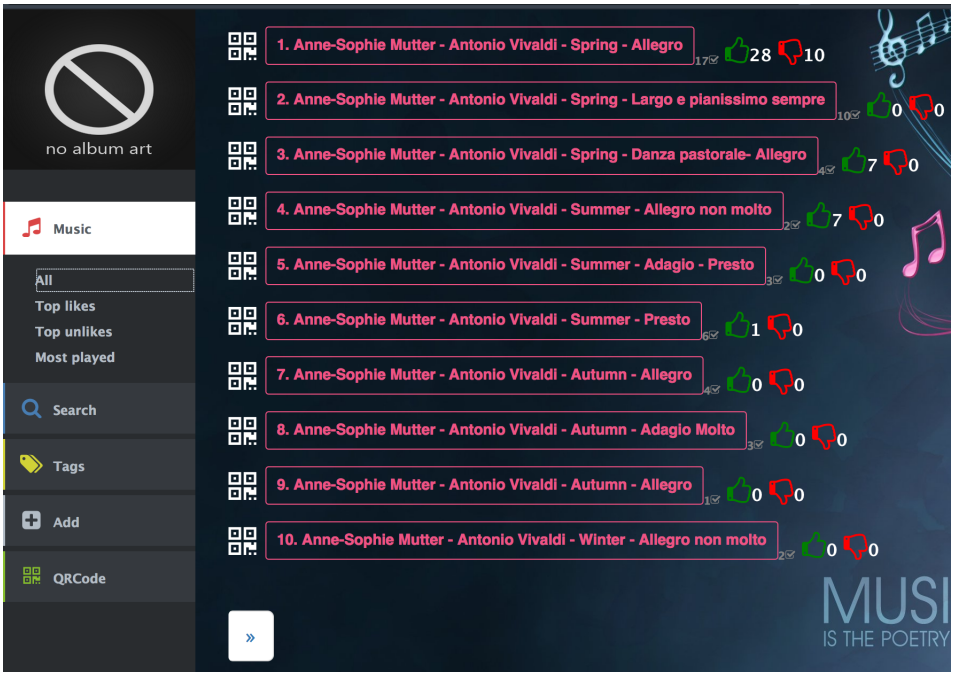


Figura 5.15: Lista de musicas.

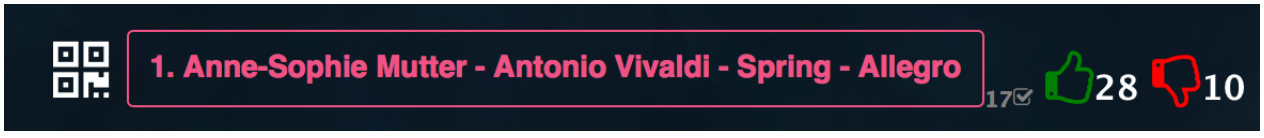


Figura 5.16: Opções de uma música.

Top Likes Ordena a lista de musicas por *likes*.

Top Unlikes Ordena a lista de musicas por *unlikes*.

Most Played Ordena a lista de musicas pelas mais reproduzidas.

O menu seguinte "Search" é de pesquisa, como pode ser visualizado na figura 5.17.

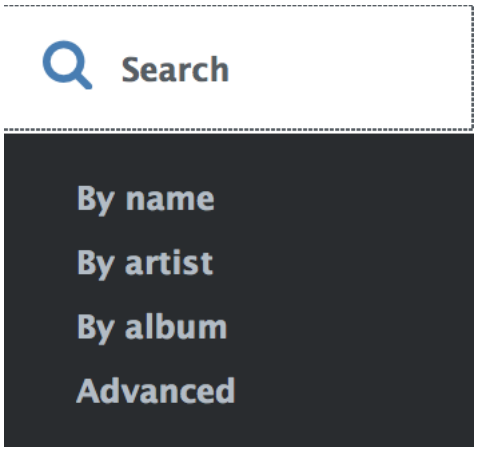


Figura 5.17: Opções de pesquisa.

Como o próprio nome de cada opção indica, este menu fornece várias opções de pesquisa.
A mais complexa, tal como a figura 5.18 indica, permite uma pesquisa composta por vários campos.

Figura 5.18: Formulário de pesquisa de musicas.

Depois de se preencher os campos que se pretendam basta clicar no botão “Search music” e se houverem resultados, irá aparecer a lista de musicas que vão de encontro com os campos introduzidos no formulário representada na figura anterior (5.18).

Outro menu disponível está relacionado com as *tags*.

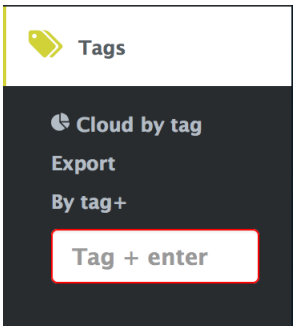


Figura 5.19: Menu de *tags*.

Este menu permite visualizar um conjunto de *tags* em forma de nuvem, pesquisar uma *tag* e exportar.
A titulo de exemplo a nuvem pode ser visualizada do seguinte modo (figura 5.20).

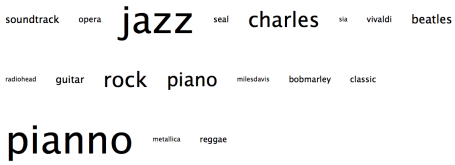


Figura 5.20: Exemplo de uma nuvem gerada.

Ao se clicar numa *tag* dentro da nuvem serão apresentadas todas as *tags* das musicas da *tag* clicada.

Em relação à pesquisa, esta pode ser realizada com uma ou várias *tags*. Para tal, é apenas necessário deixar um espaço em branco como pode ser visto na figura 5.21.

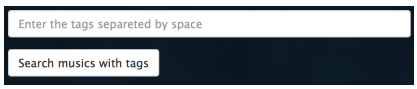


Figura 5.21: Exemplo de uma pesquisa por várias *tags*.

Para existir um conjunto de musicas é preciso que estejas sejam adicionadas ao *audio player*. Esta adição é realizada pasta a pasta, ou seja, o utizador deve agrupar numa pasta todas as musicas que deseja inserir preencher os campos que podem ser vistos na figura 5.22.

Por fim, existe o menu de leitura de QR *Codes*. Esta opção permite ler qualquer tipo de exportação de QR *Code*. Para tal, o utilizador precisa de clicar em “Read” e brandir

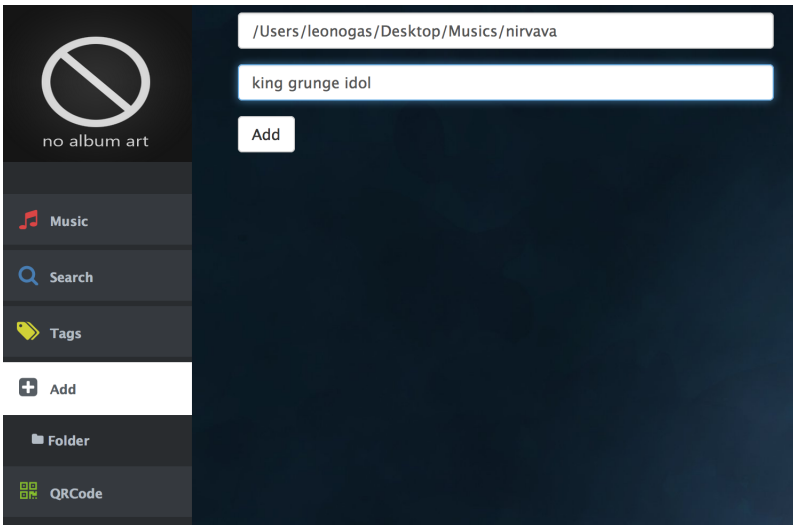


Figura 5.22: Exemplo inserção de um conjunto de musicas com *tags*.

o QR *Code* em direção *webcam* e quando este QR *Code* for reconhecido, é reproduzido uma música ou um conjunto de musicas no caso de se tratar de uma *tag*.

Utilizando o *audio player* pela linha de comandos é possível obter as mesmas funcionalidades que o *audio player* com a interface *web* menos a exportação de *tags* e a possibilidade do *audio player* reconhecer os QR *Codes*.
Esta versão permite ainda assim a reprodução de uma *tag*. Apesar de ser baseada num interface por linha de comandos, é muito simples de se utilizar porque existe um comando "help"que permite visualizar exemplos para todos os comandos disponíveis tornando assim o uso desta versão por linha de comandos muito simples e prática.
A figura seguinte (5.23) permite a visualização de todos os comandos possíveis. Estes, apresentam um exemplo prático de uso para o utilizador perceber facilmente como deve proceder. Em alguns casos aparecerão algumas notas de modo a facilitar o seu uso.

Como a interação está toda explicada neste comando "help" serão apenas descritos dois exemplos de interação com o *audio player* utilizando a linha de comandos.

Para reproduzir uma música, o utilizador deve primeiro procurar a música em questão com o comando "show next group" e depois reproduzir a música que pretender, neste caso foi a primeira, como pode ser visto na figura "playmusic". Posteriormente foi executado o comando "stop" para cancelar a reprodução da música.

Para qualquer comando que não seja reconhecido pelo *audio player* será apresentada a mensagem apresentada na figura 5.25 .

```
teste> help
Options:
    quit, Close applicacion
    play, Play a music
        ex: play 1
    random, Play a random music from the current selection
    pause, Pause the music
    stop, Stop the music
    add folder, Add complete folder to playlist
        ex: add folder path
    add, Add a music file with tags
        ex: add music_path_file tag1 tag2 ... tagN
        note: The music_path_file shouldn't have empty spaces
    remove, Remove a music
        ex: remove 1
    show previous group, Show the previous N musics
    show next group, Show the next N musics
    pagination, Set the pagination number
        ex: pagination 10
    tag music: Add a tag to music
        ex: tag music 1 tag1 tag2 ... tagN
    untag music: Remove a tag to music
        ex: untag music 1 tag1 tag2 ... tagN
    tag selection: Add a tag to selection
        ex: tag selection tag1 tag2 ... tagN
    untag selection: Remove a tag to selection
        ex: untag selection tag1 tag2 ... tagN
    select, Select playlists
        ex: select tag1 tag2 ... tagN
        note: The tag shouldn't have the character #
teste>
```

Figura 5.23: Comando “help” utilizado na versão em linha de comandos.

```
teste> show next group
1. 01_-_antonio_vivaldi_-_spring_-_allegro
2. 02_-_antonio_vivaldi_-_spring_-_largo_e_pianissimo_sempre
3. 03_-_antonio_vivaldi_-_spring_-_danza_pastorale_-_allegro
4. 04_-_antonio_vivaldi_-_summer_-_allegro_non_molto
5. 05_-_antonio_vivaldi_-_summer_-_adagio_-_presto
6. 06_-_antonio_vivaldi_-_summer_-_presto
7. 07_-_antonio_vivaldi_-_autumn_-_allegro
8. 08_-_antonio_vivaldi_-_autumn_-_adagio_molto
9. 09_-_antonio_vivaldi_-_autumn_-_allegro
10. 10_-_antonio_vivaldi_-_winter_-_allegro_non_molto
teste> play 1
teste> stop
teste>
```

Figura 5.24: Reprodução de uma música utilizando a linha de comandos.

```
teste> how previous group
Invalid command, try again!
teste> show next group
```

Figura 5.25: Comando não reconhecido pelo *audio player*.

Capítulo 6

Implementação

Nesta fase descreve-se como foi implementado o desenho apresentado no capitulo anterior (5).

Hoje em dia, existe uma grande variedade de tecnologias disponíveis que facilitam a implementação de arquiteturas e a integração entre sistemas. Por exemplo, o uso de *web services* permite a comunicação entre sistemas que foram desenvolvidos em linguagens diferentes, algo que, inicialmente, podia não estar previsto mas que é agora possível devido à possibilidade de se utilizar *web services*.
Como tal, também existe uma grande variedade de bibliotecas disponíveis tanto para Java e Javascript (apresentadas no capitulo (4.1)) como para o *front-end*. Este *front-end* ao ser uma interface *web* existe uma maior flexibilidade na apresentação de resultados porque, cada vez mais, existem bibliotecas e/ou *frameworks* que facilitam a esta construção. Neste caso, o *front-end* será visualizado num *browser* que suporte HTML 5 e Javascript. Em relação ao *back-end* foi desenvolvido em Java e suportado por um servidor *web*.

De modo a permitir extensibilidade do projeto, este, foi desenvolvido com o planeamento adequado para permitir uma rápida alteração, seja de que componente for. Para isso, foram necessários utilizar alguns padrões de desenho que garantem que as melhoras praticas disponíveis foram aplicadas. Serão apresentadas alguns dos padrões utilizados e explicando qual foi o problema que estes resolveram.

6.1 Alguns padrões de desenho

6.1.1 Singleton

Este padrão, além de de como objetivo garantir que apenas uma instância da classe alvo criada também é utilizado quando é preciso ter um único ponto de acesso global à instância da classe. Esta instância deve ser utilizada algumas vezes porque, ao estar sempre em memória, consome alguns recursos.
Aplicado ao projeto, este padrão é utilizando numa qualquer implementação de um for-

mato de música a reproduzir como por exemplo o MP3 e FLAC. Esta opção acabou por ser tomada porque sendo estas implementações únicas e sendo chamadas diversas vezes no decorrer de uma execução normal (reproduzir uma música, parar, trocar de música) facilita bastante a implementação.

6.1.2 Strategy

Este, é talvez o padrão central do projeto. Este padrão permite definir novas operações sem alterar as classes dos elementos sobre os quais opera. Reutiliza o contexto no qual se insere e permite escolher uma opção a tomar entre várias. Neste caso, a solução acabou por ficar na resposta à atribuição da camada de negócio aos pedidos enviados por parte do cliente para o servidor. Cada pedido, desde que diferente, tem a sua lógica de negócio associada e, como tal, este padrão é utilizado para decidir que delegação deve ser efetuada de modo a responder com assertividade ao pedido realizado pelo cliente.

6.1.3 Adapter

Este padrão foi bastante útil. Este padrão permite converter uma interface para outra e adaptar dados quando a interface disponível não é a mais adequada.
Tal como já foi referido, foi desenvolvida uma biblioteca (musictagid3) especifica para o projeto em que foi necessário adaptar o contrato existente, e, dessa forma, foi utilizado este padrão.
Outro exemplo da utilização deste padrão é na implementação de um tipo de música, neste caso MP3 ou FLAC, onde para a criação deste tipo é necessário adaptar os contratos existentes das bibliotecas que reproduzem o tipo musical.

6.2 NanoHTTPD

Até agora o servidor *web* nunca tinha sido apresentado. A escolha foi o NanoHTTPD. Este servidor *web* tem como função principal responder aos pedidos enviados pelo cliente. O NanoHTTPD é um servidor HTTP bastante leve que foi projetado para ser embutido em outras aplicações. Para este caso em concreto, serve perfeitamente uma vez que além de permitir esta troca de informações entre cliente e servidor, é bastante rápido e fácil de se utilizar.

6.3 *Responsive*

Com o aumento da variedade de dispositivos onde as aplicações *web* são visualizadas torna-se imprescindível adaptar o tipo de informação para cada tipo de dispositivo, seja

ele um monitor, um *tablet* ou um *smartphone*.

Seria impensável voltar a ter que desenhar e implementar múltiplas funções para vários tipos de dispositivo diferentes. Como tal, torna-se extremamente importante criar aplicações *web responsive*. Um exemplo de uma adaptação para diferentes dispositivos pode ser visto na figura seguinte (6.1), onde para cada dimensão é alterado como a informação se apresenta.

```
/* Extra small devices (phones, less than 768px) */
/* No media query since this is the default in Bootstrap */

/* Small devices (tablets, 768px and up) */
@media (min-width: @screen-sm-min) { ... }

/* Medium devices (desktops, 992px and up) */
@media (min-width: @screen-md-min) { ... }

/* Large devices (large desktops, 1200px and up) */
@media (min-width: @screen-lg-min) { ... }
```

Figura 6.1: Exemplo de uma adaptação para diferentes dispositivos

"In order to embrace designing native layouts for the web—whatever the device—we need to shed the notion that we create layouts from a canvas in. We need to flip it on its head, and create layouts from the content out." [5]
"Responsive web design offers us a way forward, finally allowing us to "design for the ebb and flow of things"." [5]

Para implementar uma página *responsive* foi utilizado Bootsrap.
Bootstrap é uma *framework* HTML, CSS e Javascript que permite desenvolver páginas *web responsive*.

"Designed for everyone, everywhere." [9]

Serão agora anotadas algumas das vantagens associadas ao uso de Bootstrap no desenvolvimento de aplicações *web responsive*:

1. Adapta o *layout* automaticamente de acordo com o tamanho da resolução onde estiver a ser executado.

2. Tem uma maior facilidade de utilização e atualização de conteúdo.

3. Permite manter um página *web* mais leve para o ambiente de execução.

4. Permite um desenvolvimento mais rápido e fácil.

5. É fortemente recomendado pela Google

Para se utilizar Bootstrap em qualquer aplicação é necessário utilizar a seguinte diretoria de ficheiros:

```
bootstrap/
├── css/
│   ├── bootstrap.css
│   ├── bootstrap.css.map
│   ├── bootstrap.min.css
│   ├── bootstrap.min.css.map
│   ├── bootstrap-theme.css
│   ├── bootstrap-theme.css.map
│   ├── bootstrap-theme.min.css
│   └── bootstrap-theme.min.css.map
├── js/
│   ├── bootstrap.js
│   └── bootstrap.min.js
└── fonts/
    ├── glyphicons-halflings-regular.eot
    ├── glyphicons-halflings-regular.svg
    ├── glyphicons-halflings-regular.ttf
    ├── glyphicons-halflings-regular.woff
    └── glyphicons-halflings-regular.woff2
```

Figura 6.2: Estrutura de ficheiros do Bootstrap.

Como pode ser visto na estrutura da figura 6.2, existem três pastas:

A pasta "css" inclui as bibliotecas necessárias para a utilização de CSS. CSS significa *Cascading Style Sheets* que é uma linguagem de folhas de estilo utilizada para definir a apresentação de documentos escritos em uma linguagem de marcação, como HTML ou XML. Tem como objetivo separar o formato e o conteúdo de um documento.
Já a pasta "js" inclui as bibliotecas necessárias para a utilização de funções Javascript disponibilizadas pelo Bootstrap.
Por fim, a pasta "fonts" permite a utilização de ícones diversificados para embelezar a página.

6.4 Geração de nuvem de *tags*

De modo obter uma ideia mais apurada sobre a geração da nuvem que foi apresentada na Análise (4.5) vai ser ilustrado um exemplo onde se pode perceber, facilmente, qual é o resultado esperado após a geração da nuvem de *tags*.

A tabela 6.1 permite perceber qual são as tags existentes e o peso que cada *tag* tem vai ter na nuvem apresentada. Como tal, e a titulo de exemplo, existem nove *tags* em que os seus pesos (número de ocorrências) aparecem na segunda coluna. Como se percebe pelos valores apresentados, vão existir *tags* com pouca expressão na nuvem, como por exemplo as tags "instrumental" e "pop" e por outro lado, com uma maior expressão, como por exemplo as tags "grunge", "rock" e "jazz".
O resultado final tem de estar de acordo com os pesos apresentados e pode ser visto logo

abaixo da tabela 6.1 na figura 6.3.

Nome da tag	Número de ocorrências em todas as musicas
rock	100
metal	50
hiphop	30
grunge	300
hardmetal	60
jazz	100
soul	30
instrumental	5
pop	10

Tabela 6.1: Tabela de ocorrências de *tags*.

Com a tabela supracitada 6.1, a nuvem gerada deverá ter um aspeto representado na figura 6.3.

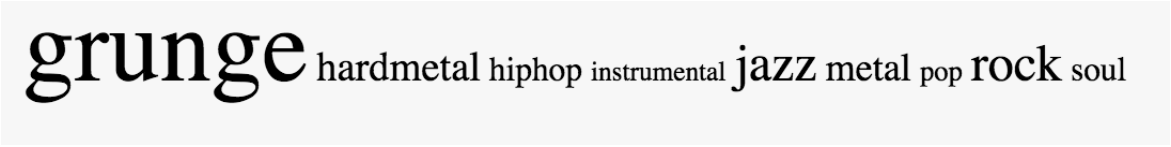


Figura 6.3: Exemplo de geração de uma nuvem.

Para se gerar este tipo de nuvem foram apresentadas as bibliotecas já referidas na Análise (4.5). Será agora detalhado de que forma se deve proceder para utilizar qualquer das bibliotecas analisadas.

6.4.1 jQCloud

Além da importação da biblioteca jQCloud e dos CSS para embelezar o *output* para gerar uma nuvem é apenas necessário utilizar uma divisão em HTML (pode ser visto na figura 6.4) e depois chamar uma função que altere o seu conteúdo (pode ser visto na figura 6.5). O resultado final pode ser visto na figura 6.6.

```
<div id="my_fcul_example" style="width: 550px; height: 350px; border: 1px solid #ccc;"></div>
```

Figura 6.4: Declaração de uma divisão em HTML

```
<script type="text/javascript">
  var word_list = [
    {text: "rock", weight: 100},
    {text: "metal", weight: 50},
    {text: "hiphop", weight: 30},
    {text: "grunge", weight: 300},
    {text: "hardmetal", weight: 60},
    {text: "jazz", weight: 100,},
    {text: "soul", weight: 30,},
    {text: "instrumental", weight: 5},
    {text: "pop", weight: 10}
  ];
  $(function() {
    $("#my_fcul_example").jQCloud(word_list);
  });
</script>
```

Figura 6.5: Chamada à função que altera o conteúdo da divisão.

Figura 6.6: Nuvem gerada utilizando a biblioteca jQCloud.

6.4.2 wordcloud2

As importações requeridas são precisamente as mesmas da biblioteca jQCloud.
A diferença existe em relação à biblioteca anterior reside na construção do *array* de *tags* e na chamada à função para gerar a nuvem

A construção da divisão em HTML pode ser igual à anterior (jQCloud) e na figura 6.7 é apresentado a forma como se procede para utilizar esta biblioteca de forma a gerar uma nuvem de palavras com base nos dados da tabela 6.1.

O resultado final desta biblioteca pode ser visto na figura 6.8.

```
var word_list = [
  ["rock", 100],
  ["metal", 50],
  ["hiphop", 30],
  ["grunge", 300],
  ["hardmetal", 60],
  ["jazz", 100],
  ["soul", 30],
  ["instrumental", 5],
  ["pop", 10]
];
$(function() {
 WordCloud(document.getElementById('my_fcul_example'), { list: word_list } );

});
```

Figura 6.7: Construção do *array* e chamada à função para gerar a nuvem.

Figura 6.8: Nuvem gerada utilizando a biblioteca wordcloud2.

6.4.3 awesomeCloud

Esta biblioteca, ao contrário da jQCloud e wordcloud2 obriga a definir alguns parâmetros (pode ser visto na figura 6.9) e também obriga a criar, dinamicamente, um elemento por cada palavra com o respetivo peso (dentro do atributo "data-weight") que também pode ser visto na figura 6.10. Nas duas bibliotecas anteriormente descritas, estes eram passados num *array*.

A figura 7.1 ilustra qual é o resultado final obtido.

```
$(function() {
    var settings = {
     "size" : {
       "grid" : 16
     },
     "options" : {
       "color" : "random-dark",
       "printMultiplier" : 3
     },
     "font" : "Futura, Helvetica, sans-serif",
     "shape" : "square"
    };
$( "#my_fcul_example" ).awesomeCloud( settings );
  });
```

Figura 6.9: Definição de alguns valores parametrizáveis.

```
<div id="my_fcul_example" style="border:1px solid #f00;height:450px;width:450px;">
  <span data-weight="100">rock</span>
  <span data-weight="50">metal</span>
  <span data-weight="30">hiphop</span>
  <span data-weight="300">grunge</span>
  <span data-weight="60">hardmetal</span>
  <span data-weight="100">jazz</span>
  <span data-weight="30">soul</span>
  <span data-weight="5">instrumental</span>
  <span data-weight="10">pop</span>
</div>
```

Figura 6.10: Criação dos elementos por palavra.

Figura 6.11: Nuvem gerada utilizando a biblioteca awesomeCloud.

Capítulo 7

Resultados

Neste capitulo, analisam-se os resultados obtidos após algumas experiências utilizando o *audio player*.

Das simulações efetuadas vai ser apresentado um caso de teste que em que foi criado o seguinte ambiente:

1. Um computador portátil, mais concretamente um MackBook Pro.
2. 6 participantes com idades entre os 25 anos até aos 31 anos.
3. Uma sala com alguma luminosidade utilizando a luz do teto ou aproveitando a luz natural se o céu estiver azul claro.
4. Cada participante pode exportar até um máximo de três QR *Codes* com qualquer tipo de exportação permitida.
5. Cada participante apenas sabe do seu "conjunto de exportação" efetuado.
6. Nenhum participante pode ter mais do que uma interação do que todos os outros participantes, ou seja, o primeiro participante tem que esperar que os outros cinco participem até pode voltar a participar novamente.

Inicialmente, foi explicado a cada um dos participantes o funcionamento geral e o modo de interação com o *audio player*. Após essa explicação, foram exportados até um máximo de três QR *Codes* por cada pessoa, onde cada exportação podia ser uma música ou um *tag*. A partir desse momento, todos os participantes quiseram desde logo reproduzir a sua exportação. A ordem acabou por ficar definida após uma atribuição aleatória e criou-se assim o ambiente ideal para começar a utilizar o *audio player*.
O primeiro passo foi perceber até que distância a *webcam* conseguia reconhecer o QR *Code*. A distancia foi facilmente identificada (até 3 metros) e achou-se aceitável mas, como é percetível, esta distancia depende da qualidade da *webcam* em questão e da luminosidade existente.

O primeiro participante a brandir um QR *Code*, escolheu reproduzir uma música. Terminada a reprodução da música, o segundo participante brandiu o seu QR *Code* e o *audio player* reproduziu de imediato a música que este participante esperava ouvir. A partir daqui, todos perceberam o *modus operandi* do *audio player* e até todos terem brandido os seus três QR *Codes* houveram alguns participantes que desejaram ter exportado mais *tags* em vez de musicas porque as *tags*, ao estarem associados a mais musicas, demoram mais tempo a reproduzir em relação a todos aqueles que exportaram apenas um música. Em suma, conclui-se que, todos os participantes sentiram-se atraídos pelo *audio player* e procuraram ter maior "tempo de antena" na reprodução de musicas.

A figura 7.1 apresenta o aspeto final do *audio player*:

Figura 7.1: *Audio player* reproduzindo uma música.

Já na figura 7.2 podem ser vistos os participantes numa festa, com algum divertimento à mistura, utilizando o *audio player* para reproduzr as suas preferências musicais utilizando para tal QR *Codes*.

Figura 7.2: Captação de um reconhecimento de um QR *Code* pelo *audio player*.

De modo a perceber o *feedback* dos intervenientes, foi efetuado um questionário a todos aqueles que participaram nos teste realizados ao *audio player*. Este questionário pode ser consultado no Apêndice C.

Os resultados vão ser diferenciados porque foram efetuados testes com diferentes etárias.

- Dos quatro aos oito - o *feedback* geral das crianças foi bastante positivo. Estas, acharam o *audio player* bastante divertido porque podiam reproduzir de uma forma

rápida e simples as músicas que tinham escolhido. Nenhuma delas sabia o que era um QR *Code* nem uma *tag*. Todas voltavam a utilizar o *audio player* e todas acharam que é de fácil utilização. Todas elas recomendavam o uso deste *audio player* aos seus amigos. A simulação foi efetuada com quatro crianças.

- Dos 25 aos 31- o *feedback* desta faixa etária foi positivo. Todos os participantes responderam de positivamente a quase tudo e inclusive deram sugestões. Todos sabiam o que era um QR *Code* e uma *tag* e gostaram imenso da ideia de se puder associar várias *tags* a diferentes musicas porque, permite assim, criar *playlists* diversificadas. Como ponto negativo, focaram-se na inserção de musicas (que é realizada por pastas) e porque o *audio player* não suporta outro tipo de formatos. A simulação foi efectuada com 6 participantes dentro da faixa etária supracitada.

- Dos 65 aos 72 - Esta foi talvez a faixa etária que mais se divertiu com o uso do *audio player*. Todos os participantes não tinham muito (ou quase nenhum) conhecimento a nível técnico , ou seja, não sabem operar em nenhum *audio player*. Por isso, tento a possibilidade de controlar o *audio player* com recurso a QR *Codes*, todos estes utilizadores gostaram bastante da ideia e ficaram algo espantados quando a musica era reproduzida. De um modo geral, gostaram da usabilidade e acharam extremamente fácil o seu uso. Nenhum dos utilizadores utiliza plataformas de entretenimento. Participaram quatro idosos na simulação.

  De um modo geral, pode-se afirmar que a ideia de utilizar QR *Codes* num ambiente festivo é interessante e segundo o *feedback* recolhido de grande parte dos participantes, o *audio player* foi muito bem conseguido.

  Podem agora ser visualizada duas fotos captadas enquanto os participantes utilizavam o *audio player*.

Figura 7.3: Participante brandindo o seu QR *Code*.

Figura 7.4: Dois participantes disputando o melhor posicionamento para se capturado o seu QR *Code*.

Capítulo 8

Trabalho futuro

Durante a realização deste trabalho, levantaram-se algumas questões pertinentes que suscitaram muito interesse, quer seja por completarem todo o trabalho que foi feito quer por permitirem, cada vez mais, diferentes cenários que ajudam a que exista uma maior interação entre os participantes e o *audio player*. Como tal, vão ser apresentados algumas linhas de orientação para trabalho futuro:

- Realizar mais testes com um número de participantes maior para perceber até que ponto quanto maior for o número de pessoas a interagir pode, ou não, aborrecer todo um conjunto de participantes.
- Utilizar um *smartphone* para leitura do QR *Code* de forma a permitir a qualquer pessoa, dentro das distancias alcançadas quer por *wireless* ou por *bluetooth*, uma maior comodidade na hora de reproduzir um QR *Code*.
- Permitir aos participantes adicionar musicas ao *audio player*. Estas musicas devem estar guardadas nos seus *smarphones*.
- Reproduzir uma maior variedade de formatos musicais além do MP3 e FLAC.
- Estudar novas formas de interagir com o *audio player* além do brandimento de QR *Codes* em direção à *webcam*.
- Desenvolver estudos relacionados com a interface de visualização. Estes estudos devem aprimorar qual será o aspeto visual que a maioria os participantes preferem.

Capítulo 9

Conclusão

Este projeto esperava uma maior aproximação e interação entre os utilizadores e o *audio player* e os resultados do projeto foram bastante animadores. Esta aproximação é, em grande parte, estimulada pela forma de interagir com o *audio player* sem que o utilizador se sinta incomodado com tal. Isto não quer dizer que, utilizando os *audio players* tradicionais, o utilizador se sinta desconfortável com o modo de interagir com este mas percebendo que, ao utilizar um *audio player* com estas características, tem a possibilidade de controlar, em parte, algum do funcionamento dum *audio player* com recurso a uma tecnologia diferente do usual e que dá algum divertimento na hora de interagir porque, desde que bem utilizada, favorece a utilização a vários possíveis utilizadores.
O utilização de QR *Codes* para interagir com o *audio player* pode ser visto como um mecanismo diferente e divertido e que permite alterar o comportamento dos participantes.
Pode-se considerar que é uma vantagem utilizar este *audio player* porque, além das enormíssimas possibilidades disponíveis a partir deste cenário como por exemplo: fazer com que um grupo de participantes adivinhe uma *tag* que escolhida, tentar adivinhar qual é o cantor de uma determinada música, entre outros, existe um maior número de pessoas a ser atraídos a participar no processo de seleção das músicas, ao invés de existir apenas uma ou duas pessoas que estavam encarregues de interagir com a tradicional interface gráfica utilizando o teclado e/ou rato para transmitir ações a serem realizadas.
De um modo geral, pode-se concluir que, utilizando este *audio player*, numa festa ou num qualquer ambiente de divertimento, as pessoas querem participar no momento de decisão das próximas musicas a serem reproduzidas e sentem maior vontade de convencer quem ainda não interagiu com o *audio player*, quer para conhecer o seu estilo musical preferido ou até mesmo por acharem este tipo de interação mais atraente e menos aborrecida.

Apêndice A

Instalação

Este apêndice tem como principal objetivo fornecer um manual de instalação do *audio player*, que tentará ser o mais simples possível.

O *audio player* é disponibilizado num ficheiro em formato zip de seu nome "Geek-MusicPlayer".

Para se proceder à configuração do *audio player* é necessário instalar o sistema de gestão de base de dados MySQL. Este ficheiro disponibilizado tem a seguinte estrutura (A.1).

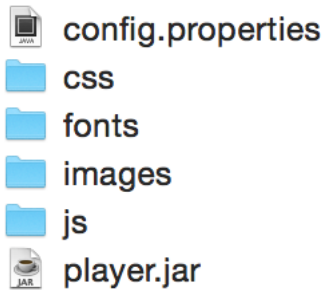


Figura A.1: Estrutura de ficheiros do *audio player*.

config.properties

Este ficheiro contem alguns parâmetros que são configuráveis e que podem variar consoante o modo de instalação do MySQL.

Assim, serão descritos todos os parâmetros que devem ser configurados:

name Este primeiro parâmetro representa o nome do *audio player*

database Este segundo parâmetro é responsável por indicar o nome da base de dados.

username Este campo indica o utilizador de acesso à base de dados.

password Este é responsável por indicar a senha de acesso do utilizador.

url Indica o caminho de instalação da base de dados.

driver Interface entre a aplicação Java e o servidor de base de dados MySQL.

dburl Indica o caminho da base de dados.

port Indica a porta de acesso à base de dados.

clientPort Indica a porta do cliente *web*.

clientUrl Indica o endereço de acesso à aplicação *web*.

css

Esta pasta contém todas as folhas de estilo em cascata (CSS) utilizados no programa.
Assim, todos as alterações a serem efetuadas a nível de CSS serão efetuadas nestes ficheiros.

fonts

Esta pasta contém todas as fontes tipográficas utilizadas.

images

Esta pasta contém apenas a imagem de fundo utilizada no *audio player*.

js

Nesta pasta, estão todas as bibliotecas Javascript utilizadas. Além disso, existem também os ficheiros de "webui.js" e "player.js" que são os ficheiros principais do *audio player*.

Após isso, será necessário executar o ficheiro "player.rar". Este ficheiro poderá ser executado da seguinte forma na linha de comandos:

```
java -jar player.jar
```

Finalmente, deveremos abrir o seguinte URL no *browser* com os seguintes parâmetros configurados no ficheiro "config.properties":

```
clientUrl + clientPort
```

Apêndice B

Personalização do *audio player*

Para alterar a imagem de fundo basta substituir o ficheiro “back.jpg” pela nova imagem de fundo. Este ficheiro encontra-se dentro da pasta “images” (apresentado aqui A.1).

Para personalizar o menu lateral do lado esquerdo (apresentado aqui 5.15), deve-se aceder ao ficheiro “sidebar-collapse.css” que se encontra dentro da pasta “cs”.
A classe definida como **sidebar-left-collapse** pode que pode ser alterada. Por exemplo, para se alterar a cor, de cinzento para qualquer outra, deve-se alterar a propriedade “background-color” desta classe.

```
.sidebar-left-collapse {
    font-family: Arial, Helvetica, sans-serif;
    position: fixed;
    top: 0;
    left: 0;
    background-color: #292c2f;
    width: 180px;
    height: 100%;
}
```

Figura B.1: Menu personalizável.

Para se personalizar uma opção escolhida dentro do menu, deve-se aceder ao mesmo ficheiro mas à classe “.sidebar-links div.selected >a”. Aqui, por exemplo, pode-se trocar o fundo quando se selecionar uma opção.

```
.sidebar-links div.selected > a{
    background-color: #ffffff;
    color: #565d63;
    line-height: 2.3;
    margin: 0;
}
```

Figura B.2: Opção dentro do menu personalizável.

Para se alterar a cor do texto de uma música da lista de musicas do *audio player* (apresentado aqui 5.15), basta aceder ao ficheiro à classe “.player h5”.

O contorno à volta da cor do texto ou o tipo de letra, do texto acima apresentado, também podem ser alterados. Para tal, é necessário alterar os valores correspondentes na classe “.player h5 a”.

```
.player h5{
    color: #F05283;
    padding: 6px 3px;
}
```

Figura B.3: Cor do texto de uma música apresentada.

```
.player h5 a{
    color:inherit;
    text-decoration: none;
    display: inline-block;
    border: 1px solid #F05283;
    padding: 10px 10px;
    border-radius: 3px;
    font: bold 14px/1 'Open Sans', sans-serif;

}
```

Figura B.4: Alteração do tipo de letra ou cor do texto.

No caso de se pretender alterar a cor do preenchimento quando se seleciona uma determinada música, é necessário apenas alterar o valor da propriedade "background-color".

```
.player h5 a:hover{
    background-color:#F05283;
    transition:0.1s;
    color:#fff;
}
```

Figura B.5: Alteração do preenchimento da região selecionada.

Apêndice C

Questionario

Plano de Testes

1. Perfil do Utilizador

Nome: _____

Zona de Habitação: _____

Sexo: Masculino Feminino (fazer um circulo à volta da sua escolha)

Idade: _____

Habilitações Literárias: _____

Profissão: _____

2. Pré-Questionário

1. Com que frequência utiliza plataformas de entretenimento?

Nunca	Pouco	Ocasionalmente	Frequentemente	Sempre

2. Caso tenha respondido nunca, ignore esta pergunta. Como classifica o seu nível de experiência na utilização deste topo de plataformas?

Nada Proficiente	Pouco Proficiente	Razoável	Proficiente	Muito Proficiente

3. Já sabia o que era um QR *Code*?

Sim	Não

4. Já sabia o que era uma *Tag*?

Sim	Não

3. Questionário

1. De um modo geral, como classifica a usabilidade do leitor de música?

Muito difícil	Difícil	Normal	Fácil	Muito fácil

2. O que tem a dizer sobre a utilização de QR *Codes* para reproduzir músicas?

Inútil	Neutro	Divertido	Muito divertido

3. Voltava a usar este leitor de música numa festa?

Sim	Não

4. Como classifica a experiencia de utilização do leitor de músicas?

Inútil	Pouco interessante	Normal	Interessante	Muito Interessante

5. Trocaria o seu leitor de músicas por este?

Sim	Não

4. Classifique o grau de dificuldade de utilização de QR *Codes* para reproduzir músicas?

Muito difícil	Difícil	Normal	Fácil	Muito Fácil

Indique se concorda muito (5) ou nada (1).

5. Relativamente à possibilidade de reprodução de uma música com recurso a um QR *Code*?

1	2	3	4	5

6. E uma tag?

1	2	3	4	5

7. Gostou do visual do leitor de músicas?

1	2	3	4	5

8. Usaria este leitor de músicas frequentemente?

1	2	3	4	5

9. Recomendaria este leitor de músicas a alguém?

1	2	3	4	5

10. Gostou do visual do leitor de músicas?

1	2	3	4	5

Bibliografia

[1] Tritonus Java Sound Implementation (www.tritonus.org/plugins.html), January 2013.

[2] Apple. ios (http://www.apple.com/pt/ios/), Juve 2007.

[3] SRI International Apple Inc. Siri (https://support.apple.com/pt-pt/HT4992), October 2011.

[4] Dan Becker. Java Sound API (www.javaworld.com/javaworld/jw-11-2000/jw-1103-mp3.html), January 2013.

[5] Mark Boulton. A richer canvas, March 2011.

[6] Eric Farng. Java ID3 Tag Library (http://javamusictag.sourceforge.net/description.htm), January 2013.

[7] Google Inc. Google Play (https://play.google.com/), February 2013.

[8] Droidla Limited. QR Code (http://qrdroid.com/), February 2013.

[9] MIT. Mit (http://getbootstrap.com/), Juve 2014.

[10] Michael Patricios. MP3agic (https://github.com/mpatric/mp3agic), January 2013.

[11] Philippe Perez. QR Code Scanner (https://play.google.com/store/apps/details?id=com.pp040773.androidapps.qrviewer), February 2013.

[12] Inc. Scan. Scan (https://play.google.com/store/apps/details?id=me.scan.android.client), February 2013.

[13] Paul Taylor. JAudioTagger (www.jthink.net/jaudiotagger/index.jsp), January 2013.

[14] WB Development Team. QR Bar Code Scanner (https://play.google.com/store/apps/details?id=appinventor.ai_progetto2003.SCAN&hl=pt_BR), February 2013.

[15] Denso Wave. ZXing (`https://code.google.com/p/zxing/`), February 2013.